Fundamentals of Computer Vision

Computer vision has widespread and growing application including robotics, autonomous vehicles, medical imaging and diagnosis, surveillance, video analysis, and even tracking for sports analysis. This book equips the reader with crucial mathematical and algorithmic tools to develop a thorough understanding of the underlying components of any complete computer vision system and to design such systems. These components include identifying local features such as corners or edges in the presence of noise, edge-preserving smoothing, connected component labeling, stereopsis, thresholding, clustering, segmentation, and describing and matching both shapes and scenes.

The extensive examples include photographs of faces, cartoons, animal footprints, and angiograms. Each chapter concludes with homework exercises and suggested projects.

Intended for advanced undergraduate and beginning graduate students, the text will also be of use to practitioners and researchers in a range of applications.

Wesley E. Snyder is Professor Emeritus of Electrical and Computer Engineering at North Carolina State University. He was previously a professor at the Bowman Gray School of Medicine and worked at the GE Corporate Research and Development Center, NASA Langley Research Center, and the West German Air and Space Agency (DLR). He has published 179 research papers and was chosen as an Outstanding Engineering Educator in North Carolina in 1993. He was made fellow of the IEEE, Gladden Fellow (University of Western Australia), and Fellow of the American Institute of Medical and Biomedical Engineering.

Hairong Qi is the Gonzalez Family Professor of Electrical Engineering and Computer Science at the University of Tennessee, Knoxville. Her research interests include collaborative signal and image processing, hyperspectral imaging, and bioinformatics. She is the recipient of an NSF CAREER Award, numerous best paper awards at international conferences, and was awarded the highest impact paper from the IEEE Geoscience and Remote Sensing Society in 2012.

Fundamentals of Computer Vision

WESLEY E. SNYDER
North Carolina State University

HAIRONG QI
University of Tennessee

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

4843/24, 2nd Floor, Ansari Road, Daryaganj, Delhi - 110002, India

79 Anson Road, #06-04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107184886
DOI: 10.1017/9781316882641

First published 2017

Printed in the United Kingdom with Clays, St Ives plc

A catalogue record for this publication is available from the British Library.

ISBN 978-1-107-18488-6 Hardback

This book is dedicated to my wonderful wife of 48 years (at the moment), Rosalyn. She was patient enough to apply her editing skills to multiple passes through this book. But even more important, she was patient enough to put up with living with me.

WES

To QiQi and YangYang, my two wonderful children who probably haven't come to understand what Mom is doing, but helped nonetheless in a big way.

HQ

Contents

Part III Image Understanding

Preface

This book introduces the fundamental principles of Computer Vision to the advanced undergraduate or first-year graduate student in mathematics, computer science, or engineering.

The book is deliberately informal. The authors attempt to keep the student interested and motivated to continue reading. The student is often addressed directly, as if the student and the authors were in a classroom together. The style is somewhat casual, the authors use the first person frequently, the passive voice is seldom used, and the occasional joke may appear.

The foundations described in the title of this book take two forms: mathematical principles and algorithmic concepts. The principles and concepts are taught together, by describing a computer vision problem, e.g., segmentation, describing an algorithm that could solve that problem, and explaining the mathematical principles underlying the algorithm.

These mathematical principles include

Linear Operators Taught through a variety of applications, including:
- ***Basis Functions*** Taught through edge detectors
- ***Gaussian Convolution*** Taught through development of Gaussian edge kernels
- ***Constrained Optimization*** Taught through finding optimal edge detection kernels and through principal components
- ***The Pseudoinverse*** Taught through explaining photometric stereo
- ***Scale*** Taught through development of Gaussian edge kernels.

Nonlinear Operators Taught through mathematical morphology.

Effects of Sampling Taught by illustrating the use of small kernels for determining orientation.

Use of Optimization Taught through development of noise removal algorithms, adaptive contours for segmentation and graph cuts.

Use of Consistency Taught through Hough-like algorithms, through shape matching, and projection onto a manifold.

Projective Geometry Taught through shape from motion and shape from X algorithms.

These concepts are organized in a progression of levels, moving from pixel-level operations such as noise removal, through edge detection, segmentation, and shape description, and finally to recognition. At each level in the progression, the student learns what specific terminology means, what a good application of the concept to an image does, and one or more approaches to solving the application problem.

Almost all the images used as examples in the book are available for download. When one of these figures is used, the figure caption gives the name of the image.

What is not included in this book:
The disciplines of statistical pattern recognition and artificial neural networks [0.1] receive only passing mention in this book. This is because both of those disciplines are sufficiently important and sufficiently broad to justify entire courses in themselves.

Also not included are some topics that are of current active research interest. Deep Learning [0.3], for example, is turning out to be of great importance to the eventual success of Computer Vision systems, but such topics are better suited to an advanced topics course, after the students have been introduced to Computer Vision using this book.

We recommend that prior to this book, students have adequate knowledge of image processing [0.2], such that they learn what a pixel is, what the difference is between enhancement and restoration, how to process a color or multi-spectral image, and how to do image processing, preferably in both the spatial domain and the frequency domain, as well as the different filtering approaches.

The student using this book will receive a broad introduction to the field of Computer Vision, which includes reinforcement of many of the requisite mathematical principles.

The book is in four parts:

Part I, Preliminaries touches on biological vision, math, and implementation of algorithms in software.

Part II, Preprocessing discusses the need to "clean up" images before we start the Computer Vision algorithms by removing corruptions due to noise and blur. The need for local area operations such as convolution are included.

Part III, Image Understanding covers segmentation of an image into meaningful regions and representing those regions. Matching of both regions and scenes is included in this part.

Part IV, The 2D Image in a 3D World describes how images of objects may be related to those objects in the observed world.

Bibliography

[0.1] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley Interscience, 2nd edition, 2000.

[0.2] R. Gonzalez, R. Woods, and L. Eddins. *Digital Image Processing Using MATLAB*. McGraw-Hill, 2nd edition, 2016.

[0.3] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553), 2015.

For the Instructor

This is a first course in Computer Vision. It is intended for seniors or first-level graduate students in math-intensive curricula, such as electrical engineering, computer engineering, math, physics, or computer science.

While this book provides a "foundation" in computer vision, a foundation in university-level mathematics is required for this book. We suggest that all students have the usual three-semester course in differential calculus, at least one course in differential equations, and one or two courses in concepts of matrices and linear algebra.

The authors feel strongly that a student in Computer Vision should have a reasonably good grasp of what the computer is actually doing – and that means programming. While MATLAB is a fantastic research tool, and provides wonderful rendering tools, using MATLAB may prevent the student from gaining this insight into the basic operations (unless, of course, the student actually writes MATLAB code and doesn't use a toolbox). We prefer the student learn and use C or C++.

We have a software package available with which we have had considerable success, described in Appendix C. It can be downloaded for free for MacOSX64, Linux64, and most versions of Windows, along with a lot of images, from `www.cambridge.org/9781107184886`. This package is, however, not required to use the book. The instructor may choose any software platform, including MATLAB.

One style of teaching this course, the one we usually follow, is to have a number of small projects, roughly a week or two in duration, instead of a single end-of-class project.

In teaching this course, the principal problems we have encountered are with students who don't know the difference between a compiler and a fishing rod, and students who think an eigenvalue is made with sugar and served with syrup. The emphasis on knowing linear algebra is very important.

We also recommend that prior to taking this course, students take a class in image processing, in which they learn what a pixel is, what restoration does, and the difference between enhancement and filtering.

Our teaching style in this book is deliberately informal. We try to instill in the student the feeling that s/he is in a personal conversation with the book. For that reason, we often use the word "we" rather than the passive voice. We also include the occasional joke or silliness. The occasional smile hopefully will motivate the student to read on.

The authors of this book also wrote an earlier book on computer vision, published by Cambridge University Press in 2004. This text does contain a very small amount of material taken directly from that book.

Part I

Preliminaries

In this part of the book, we start getting ready to do Computer Vision (CV). First, we briefly introduce biological vision and the biological motivation of Computer Vision. This is followed by a chapter describing how to write computer programs to process images. We then provide a chapter of math. The emphasis in that chapter is linear algebra, as that is the mathematical discipline most used in CV. We also touch briefly on probability and function minimization.

After we have the human prepared by taking the prerequisites and reviewing the math, we need for that human to know what an image actually is. It turns out that the word "image" has a number of different meanings in different contexts, and Chapter 4 discusses them.

1 Computer Vision, Some Definitions, and Some History

No object is mysterious. The mystery is your eye.
– Elizabeth Bowen

1.1 Introduction

There are two fundamentally different philosophies concerning understanding the brain. (1) Understand the brain first. If we can understand how the brain works, we can build smart machines. (2) Using any technique we can think of, make a smart machine. If we can accomplish that, it will give us some hints about how the brain works. This book is all about the second approach, although it draws from current understanding of biological computing. In this chapter, however, we define a few terms, introduce the greater local-global problem, and then give a very brief introduction to the function of the mammalian brain.

- (Section 1.2) From signal and systems perspective, we describe the differences between Computer Vision and some other closely related fields of studies, including, e.g., image processing and pattern recognition.
- (Section 1.3) Since almost all problems in Computer Vision involve the issue of localness versus globalness, we briefly explain the "local-global" problem and the "consistency" principle used to solve this problem.
- (Section 1.4) Computer Vision is deep-rooted in biological vision. Therefore, in this section, we discuss the biological motivation of Computer Vision and some amazing discoveries from the study of the human visual system.

1.2 Some Definitions

Computer Vision is the process whereby a machine, usually a digital computer, automatically processes an image and reports "what is in the image." That is, it recognizes the content of the image. For example, the content may be a machined part, and the objective may be not only to locate the part but to inspect it as well.

Students tend to get confused by other terms that often appear in the literature, such as *Image Processing*, *Machine Vision*, *Image Understanding*, and *Pattern Recognition*.

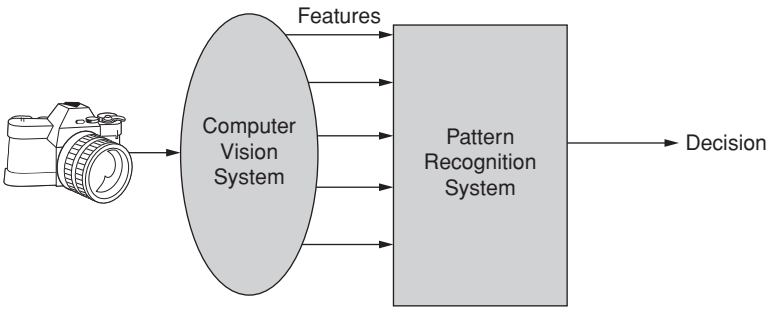


Figure 1.1 A Computer Vision system accepts an input that is an image and produces a number of measurements extracted from the image. These measurements are called *features* and are provided to a pattern recognition system that makes a decision about the object or objects being imaged.

We can divide the entire process of Image Processing into *Low-Level Image Processing* and *High-Level Image Processing*. If we interpret these processes from signal and systems perspective, it is more clear to describe their difference and similarity from the format of input/output of the system. When a *Low-Level Image Processing* system processes an input image, the output is still an image, but a somewhat different image. For example, it may be an image with noise removed, an image that does not take as much storage space as the input image, an image that is sharper than the input image, etc. Although low-level image processing does not give information about the object being imaged, it is likely to be necessary, as it enables the higher-level algorithms to work properly.

The terms *Machine Vision*, *Image Understanding*, and *High-Level Image Processing* are often used to denote Computer Vision. They are processes that operate on an input image, but the output is not an image anymore; instead, it is an interpretation of the image.

Pattern Recognition indicates an essential component of a Computer Vision system with the input being a description of the image (e.g., features that describe the object in the image) in a structured format (e.g., a vector) and the output being a recognition result based on those input descriptions. Hence, for the process of Pattern Recognition, neither the input nor the output is an image.

Figure 1.1 illustrates one interpretation of a Computer Vision system. The decision of what an object is, its *class*, is made by a Pattern Recognition (PR) system. The decision made by the Pattern Recognition system is based on measurements made on the input. These measurements are called *features*. For example, the PR system must decide whether an object is an axe or a hatchet, and it receives as input the length of the handle and the mass of the head. If the length of the handle is to be determined from an image, the system that makes that determination is a Computer Vision system.

So a Computer Vision system makes measurements in an image and reports those measurements to a pattern recognition system. If the Computer Vision system is sophisticated and returns "number of elephants in the room = 1," the PR system may have nothing to do.

There are exceptions to the definitions we just presented. Probably the best current example is the current research in *deep learning.* These systems use neural-motivated computing structures to identify specific objects in an image. One could argue that these systems are

pattern classifiers, but the problems they solve are clearly Computer Vision problems, so it would also be appropriate to classify them as Computer Vision systems [1.4].

1.3 The Local-Global Problem

Almost all problems in Computer Vision can be described as different versions of the "local-global" or "gestalt" problem. We describe it through two examples:

The first is the old parable of the blind men and the elephant: "blind men were asked to determine what an elephant looked like by feeling different parts of the elephant's body. The blind man who feels a leg says the elephant is like a pillar; the one who feels the tail says the elephant is like a rope; the one who feels the trunk says the elephant is like a tree branch; the one who feels the ear says the elephant is like a hand fan; the one who feels the belly says the elephant is like a wall; and the one who feels the tusk says the elephant is like a solid pipe."[1]

There is a visual example of the same story: Find a magazine with an interesting cover, and cut a piece of blank paper roughly twice the height and width of the magazine. Cut a small (1 or 2 cm diameter) hole in the center of the sheet and lay the paper down, covering the magazine. Now, find a friend who has not previously seen the magazine. Allow him/her to move the paper in any way at all, but constrained to only expose the magazine cover through the hole. Ask your friend to describe the cover. Chances are, you will get a description that is totally incorrect.

This is what we ask computers to do; make local measurements (e.g., gradients) all over the image, and somehow the computer infers from all those local measurements that the image is a picture of an elephant.[2]

Partial solutions to this problem are many and varied, a number of which are presented in this book. Most of them make use of *consistency*. That is, if the computer can discover that a number of measurements are consistent in some way, it may be able to infer the aspect of the global image that causes that consistency.

As you go through the material, keep in mind how what you are reading can be used to solve this greater problem.

1.4 Biological Vision

Biological computation (neurons) and biological vision (using those neurons) is the inspiration of Computer Vision. Since the start of Computer Vision as a discipline, there have been attempts to measure biological vision and model it in computers. Over the history of Computer Vision, many techniques have been developed to make use of biologically motivated models. In recent years new techniques such as *deep learning* have become popular, and those methods owe their form to early works in biological vision.

[1] From Wikipedia.

[2] We are asking the computer to solve a *gestalt* problem, discovering the identity of the content of an image from lots of locally distributed measurements.

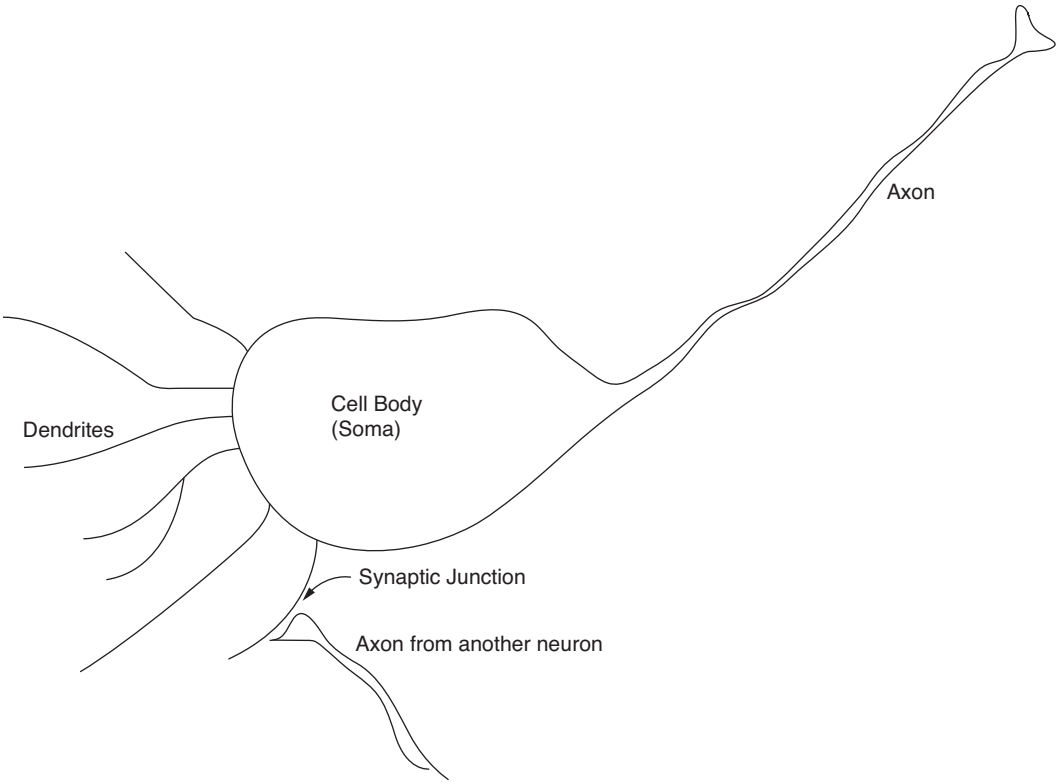


Figure 1.2 The biological neuron has many inputs (i.e., dendrites), each weighted, and a single output (i.e., axon), which may divide and go to several places.

1.4.1 The Biological Motivation

First, we consider the biological neuron, as illustrated in Figure 1.2. The *dendrites* are tiny fibers that carry signals to the neuron cell body. The cell body serves to integrate (over dendritic inputs and time) the inputs from those dendrites. This integration process is actually the accumulation of charge in the form of positive ions. When a sufficient amount of charge has accumulated, the cell membrane breaks down, allowing a massive ionic flow that propagates down the single output of the cell, the *axon*. Axons may be quite long, even as long as over 10 cm. This propagation of ion flow is referred to as an *action potential*. Information is encoded by the rate of occurrence of action potentials. A typical action potential has an amplitude in the tens of millivolts and a duration in the milliseconds.

Output/input signal transitions occur where an axon impinges on a dendrite or on a cell body. Such a junction is called a *synaptic junction*. As electrical charges move down an axon, they cause the release of a *neurotransmitter* chemical across the synaptic junction. As many as 1,000 synapses may provide input to a single neuron.

Figure 1.3 illustrates graphically two axons making connection to a single cell body through two different sized synapses. Neurotransmitters are chemicals released from the axon into the synaptic junction, and they may encourage the cell to fire (excitatory

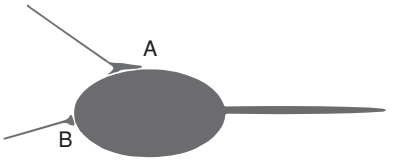


Figure 1.3 Two axons terminate on the same cell body through synapses that vary significantly. A pulse passing through synapse A is more likely to cause the postsynaptic neuron to fire than a pulse through neuron B.

synapses) or inhibit cell firing (inhibitor neurotransmitters at inhibitory synapses). In the figure, the larger synapse, A, is capable of releasing more neurotransmitter and therefore exerting more influence on the postsynaptic neuron. Changes in the ability of synapses to transmit signals appear to be one form of what we call *memory*. The influence of presynaptic neurons on postsynaptic neurons may be modelled as plus or minus sign on weights to a summer. It has been estimated that an adult nervous system possesses 10^{10} neurons. With 1,000 synapses per neuron, and 8 bits of storage per synapse (actually, synapses are not that precise . . . 5 bits is more likely), we come up with 10 terabytes of storage in your brain! So use it! Read the remainder of this chapter and beyond!

The neuron may be modeled mathematically with a sum-of-products followed by a nonlinearity, as illustrated in Figure 1.4. Therefore, the equation of the neuron can be written as

$$y_q = S_q\left(\sum_i x_i w_{iq}\right) \tag{1.1}$$

where i and q are the indices for inputs and neurons, respectively, and S_q is the nonlinear function. The nonlinearity S_q has the form of a sigmoid (having the shape of the letter "S"). A common sigmoid is the logistic function $S(x) = \frac{1}{1+e^{-ax}}$, where a controls the slope of the sigmoid in its center.

Historically, the first nonlinearity adopted in such mathematical neural network models was the simple threshold. This was done since there is clearly a threshold-like phenomenon

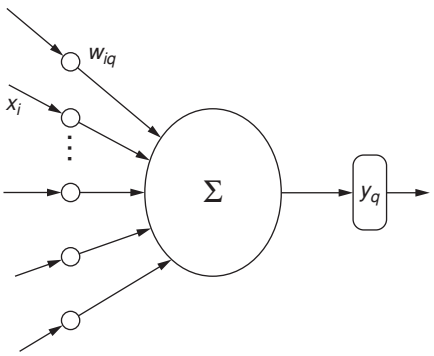


Figure 1.4 Architecture of a single neuron. Inputs are multiplied by weights before being summed. The output of the summation is passed through a threshold-like nonlinearity.

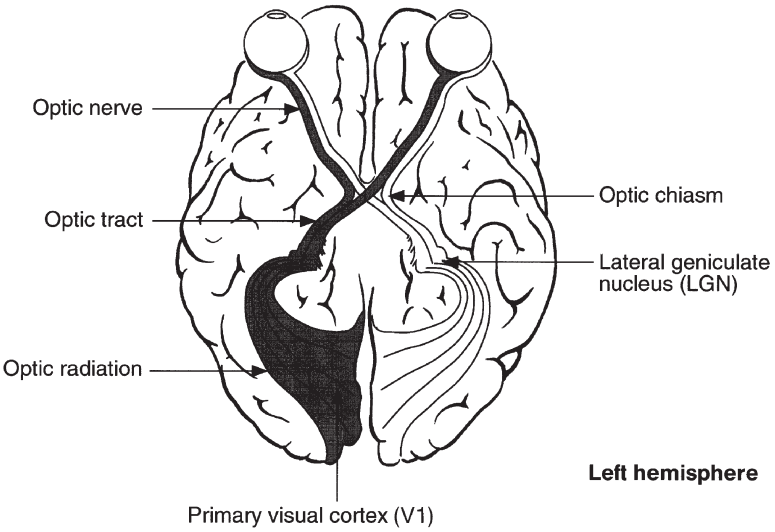


Figure 1.5 The primary visual center, V1, is at the back of the brain. From [1.7].

measurable in biological neurons. This first model, the McCulloch-Pitts neuron, was developed in 1943 [1.2]. The application of collections of such neurons led to the development of Perceptrons [1.5], which are simple linear machines with thresholds. By adding layers to a single layer machine, arbitrarily complex decision boundaries can be specified [1.3].

The use of a threshold causes a difficulty when trying to determine the weight values required to solve a particular problem. Because the threshold function is not differentiable, it becomes challenging to find the weights for multilayer networks.This problem is solved [1.6, 1.8] by replacing the threshold with a sigmoid, which is differentiable. This one observation, that a non-differentiable function may be approximated by a differentiable one, was the source of the revival of field of neural networks research.

1.4.2 Visual Perception

You already know that images are formed on the retina by light. The electrical signals generated by light sensors pass through the optic nerve, the lateral geniculate nucleus (LGN), and terminate in the visual cortex, a large area in the back of the brain, as shown in Figure 1.5. The visual cortex may actually be divided into several sections, V1 (or primary), V2, V3, V4, and V5. Images receive different processing in different areas. The visual cortex is also referred to as the *striate cortex* because it appears to have long, fairly straight bands (Figure 1.6).

The visual cortex is *retinotopic*. That is, a particular cell in the visual cortex fires when a particular light sensor in the retina is stimulated, and furthermore, if two points in the retina are close together, then the corresponding points in the visual cortex are also close together.

Mathematical modeling of neurons encountered the area of vision in 1962 with the work of Hubel and Wiesel in [1.1]. These two neurophysiologists took mammals (initially cats,

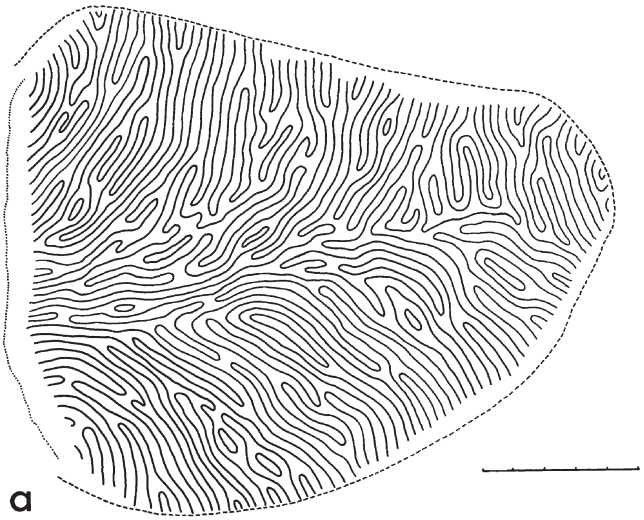


Figure 1.6 The visual cortex, when stained appropriately, seems to be divided into long bands. From [1.7].

and later monkeys) and implanted electronic probes so small that the firing of single neurons could be detected. With this capability, they made some amazing discoveries.[3]

The first discovery is that there is a correspondence between certain cells in the cortex and the presence of, not just light, but edges in the retina. Even more interesting, there are cells in the cortex that respond when edges *of a certain orientation* are present in a particular place in the retina!

Every cortical cell that fires in this way is said to have a *receptive field*, which consists of all the light receptors that, when stimulated, have some direct effect on this cortical cell. The simplest receptive fields, actually found in the retinal ganglial cells, are called "center-on" cells, illustrated in Figure 1.7, which fire when the center of the receptive field is stimulated and the surrounding is not. More interesting cells in the cortex have been found to fire when the center is stimulated by an edge with a particular orientation. That there are cells that behave in this way motivates the use of kernel operators in Chapter 5 of this book. Also found are cells that respond to motion of edges.

Another observation by Hubel and Wiesel is that the cortex has bands of neurons that receive input from the left eye, and each of these bands borders a band that receives input from the right eye. This suggests that the wiring of the cortex performs some of the computations required for humans to perceive depth.

Finally, later exploration of brain physiology has shown that as one moves from the primary visual cortex, to V2, V3, and further, the receptive fields seem to grow, suggesting a hierarchical architecture with several layers of computation and interconnection.

There is a great deal of debate and research going on in visual perception currently, and this field can be expected to continue to be very active for many years.

[3] Yes, they *did* receive a Nobel prize in 1981.

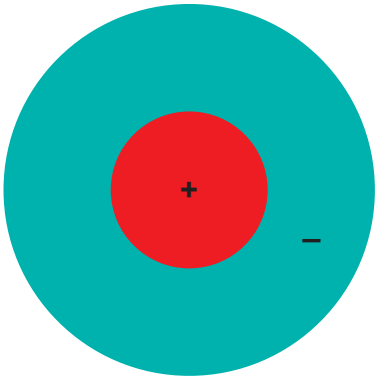


Figure 1.7 Some cells in the primary visual cortex only fire when the sum of the interior cells minus the sum of the exterior cells exceeds some threshold.

Bibliography

[1.1] D. Hubel and T. Wiesel. Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology (London)*, 160, 1962.

[1.2] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 1943.

[1.3] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

[1.4] R. Ranjan, V. Patel, and R. Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2016.

[1.5] F. Rosenblatt. The Perceptron – a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

[1.6] D. Rumelhart. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press, 1982.

[1.7] M. Tovée. *An Introduction to the Visual System*. Cambridge University Press, 2008.

[1.8] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis.

2 Writing Programs to Process Images

Computer Science is not about computers any more than astronomy is about telescopes.
– E.W. Dijkstra

2.1 Introduction

One may take two approaches to writing software for image analysis, depending on what one is required to optimize. One may write in a style that optimizes/minimizes programmer time, or one may write to minimize computer time. In this course, computer time will not be a concern (at least not usually), but your time will be far more valuable. For that reason, we want to follow a programming philosophy that produces correct, operational code in a minimal amount of programmer time. The programming assignments in this book are specified to be written in C or C++, rather than in MATLAB or JAVA. This is a conscious and deliberate decision. MATLAB in particular hides many of the details of data structures and data manipulation from the user. Most of the time, that's a good thing. However, in the course of teaching variations of this course for many years, the authors have found that many of those details are precisely the details that students need to grasp in order to effectively understand what image processing (particularly at the pixel level) is all about.

In this book, at least initially, we want the students to write code that works at the pixel level, so they come to understand what the computer is really doing. Later in the course, the student will slowly move up in levels of abstraction. We point the reader to [2.2, 2.1] for texts that emphasize the use of MATLAB.

2.2 Basic Programming Structure for Image Processing

Images may be thought of as two- or three-dimensional arrays. They are usually processed pixel-by-pixel in a raster scan. In order to manipulate an image, two- or three-nested for-loops is the most commonly used programming structure, as shown in Figures 2.1 and 2.2.

In these examples, we use two or three integers (`row`, `col`, and `frame`) as the indices to the row, column, and frame of the image. By increasing `row`, `col`, and `frame` with a step

```
.....
int row, col;
......
for (row = 0; row < 128; row++)
{
    for(col = 0; col < 128; col++)
    {
        /* pixel processing */
        ......
    }
    ......
}
```

Figure 2.1 Basic programming structure: two-nested for-loops.

one, we are actually scanning the image pixel-wise from left to right, top to bottom, frame by frame.

2.3 Good Programming Styles

Always follow one important programming construct: all programs should be written so that they will work for any size image. You are not required to write your programs so that they will work for any number of dimensions (although that is also possible), or any data type; only size. One implication of this requirement is that you cannot declare a static array and copy all your data into that array (which novice programmers like to do). Instead, you must use the image access subroutines.

Another important programming guideline is that, except in rare instances, global variables are forbidden. A global variable is a variable that is declared outside of a subroutine.

```
......
int row, col, frame;
......
for (frame = 0; frame < 224; frame++)
{
    for (row = 0; row < 128; row++)
    {
        for (col = 0; col < 128; col++)
        {
            /* pixel processing */
            ......
        }
        ......
    }
    ......
}
```

Figure 2.2 Basic programming structure: three-nested for-loops.

Use of such globals is poor programming practice, and arguably causes more bugs than anything else in all of programming. Good structured programming practice requires that everything a subroutine needs to know is included in its argument list.[1]

Following these simple programming guidelines will allow you to write general-purpose code more easily and efficiently, and with fewer bugs. As you become more skilled you can take advantage of the pointer manipulation capabilities to increase the run speed of your programs if this is needed later.

Adding some space (e.g., a blank line) between different segments of your code also improves readability. We emphasize the importance of the comments. However, do not add too many comments, since that will break the flow of your code. In general, you should add a block comment at the top of each function implementation, including descriptions of what the function does, who wrote this function, how to call this function, and what the function returns. You should also add a description for each variable declaration.

2.4 Especially Important Points for Computer Vision

If you are taking the time to read this section, you are several days of labor ahead of your friends. They will make the common errors described here, and take a *very* long time to find them. So here are a few things to avoid.

- If you read in an image, using almost any image read function, you will be reading a data set composed of integers, 8 bits each, and therefore having values which range from 0 to 255. Remember that.
- The fact of the last bullet causes problems because with 8-bit integers, $250 + 8 = 2$ and $3 - 5 = 254$.
- If you read in an image, *immediately* convert it to float. Do all your operations in float. This includes brightness scaling. Don't convert back to integer until just before you write the output image. MATLAB does this. Everything is a floating point number unless you work hard to create integers.

2.5 Image Analysis Software Toolkits

There are numerous basic computing functions that the instructor may feel are not relevant to students in a Computer Vision class, e.g., using images with different data types (float is particularly important), compressing data in files, representing 3D, 4D, or color images, etc. For these capabilities, it may be convenient to make use of an Image Analysis toolkit of some sort. There are at least two that the authors have used that serve this need: OpenCV and IFS. We use IFS (image file system) partly because we developed it, and partly because we think it has superior capabilities. The IFS system is described in Appendix C along with some example programs.

[1] C++ users be careful: The variables of a class are available to any method in that class and it is very easy to write methods which have side-effects which are hard to debug. This is exactly what we are trying to avoid with a no globals policy.

```
CC = g++
INCLUDES = -I../include
LIBS = -L../lib -lifs
CFLAGS = $(INCLUDES)

myprog: myprog.o
    $(CC) -o myprog myprog.o $(LIBS)
myprog.o: myprog.c
    $(CC) -c myprog.c $(INCLUDES)
```

Figure 2.3 An example UNIX makefile.

2.6 Makefiles

Basically, a makefile specifies how to build your project, as illustrated by the example makefile in Figure 2.3. You should use makefiles because they are far superior to just typing commands. If you are doing your software development using Microsoft Visual Studio, Mac Xcode, or some other development environment, then the makefiles are somewhat hidden from you, but it is helpful to know how they operate.

The example in Figure 2.3 is just about as simple a makefile as one can write. It states that the executable named `myprog` depends on only one thing, the object module `myprog.o`. It then shows how to make `myprog` from `myprog.o` and the library. Similarly, `myprog.o` is made by compiling (but not linking) the source file, `myprog.c`, utilizing header files found in an "include" directory. Note: to specify a library, as in the link step, one must specify the library name (e.g., `libifs.a`), but to specify an include file (e.g., `ifs.h`), one specifies only the directory in which that file is located, since the file name was given in the `#include` preprocessor directive.

In WIN64 the makefiles look like the example shown in Figure 2.4. Here, many of the symbolic definition capabilities of the make program are demonstrated, and the location of the compiler is explicitly specified.

```
CFLAGS = -Ic:\lcc\include -g2 -ansic
CC = c:\lcc\bin\lcc.exe
LINKER = c:\lcc\bin\lcclnk.exe
DIST = c:\ece763\myprog\lcc\
OBJS = c:\ece763\myprog\objs\
LIBS = ifs.lib -lm
# Build myprog.c
myprog:
    $(CC) -c $(CFLAGS) c:\ece763\myprog\mysubroutine1.c
    $(CC) -c $(CFLAGS) c:\ece763\myprog\mysubroutine2.c
    $(CC) -c $(CFLAGS) c:\ece763\myprog\myprog.c
    $(LINKER) -subsystem console -o myprog.exe myprog.obj mysubroutine1.obj
        mysubroutine2.obj $(LIBS)
```

Figure 2.4 An example WIN32 makefile.

The programs generated by IFS are (with the exception of ifsTool) console-based. That is, you need to run them inside an MSDOS window on the PC, inside a terminal window under Linux or on the Mac, using OS-X.

2.7 Assignments

Assignment 2.1: Learn how to use IFS. The intent of this assignment is to get you to use the computer and to begin to write programs.

1. Use `ifsTool` to view the following images: `imagesecho1`, `imagesecho2`, `imagesecho3`.
2. Written assignment, to be handed in: Describe in one paragraph what the three "echo" images look like. We recognize that you do not know the anatomy. Just describe what you see; include a printout of the image. If it happens that you do know the anatomy, your instructor will be impressed.
3. Programming assignment, to be handed in: Write a progam that will take in one of the above images as input and produce an output image, of the same type and the same size, but with each pixel intensity changed from `inimg(row,col)` to `outimg(row,col) = a` $\times$ `inimg(row,col) + b`, where `a` and `b` are input parameters. Try to write the program in a way that you can change the values of `a` and `b` without having to modify the code itself.
4. What is the functionality of the above program?

Bibliography

[2.1] P. Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. Springer, 2011.

[2.2] R. Gonzalez, R. Woods, and L. Eddins. *Digital Image Processing Using MATLAB*. McGraw-Hill, 2nd edition, 2016.

3 Review of Mathematical Principles

Practical problems require good math.
– R. Chellappa

3.1 Introduction

This chapter is a review of several of the topics that are prerequisite for use of this book as a text. The student should have had an undergraduate calculus experience equivalent to about three semesters and some exposure to differential equations and partial differential equations. The student should have coursework containing concepts from probability and statistics, including prior probabilities, conditional probability, Bayes' rule, and expectations. Finally, and very important, the student should have strong undergraduate-level training in linear algebra.

This chapter reviews and refreshes many of the concepts in those courses, but only as a review, not as a presentation of totally new material.

- (Section 3.2) We briefly review important concepts in linear algebra, including various vector and matrix operations, the derivative operators, eigendecomposition, and its relationship to singular value decomposition.
- (Section 3.3) Since almost all Computer Vision topics can be formulated as minimization problems, in this section, we briefly introduce function minimization, and discuss gradient descent and simulated annealing, the two minimization techniques that can lead to local and global minima, respectively.
- (Section 3.4) In Computer Vision, we are often interested in the probability of certain measurement occurring. In this section, we briefly review concepts like probability density functions and probability distribution functions.

3.2 A Brief Review of Linear Algebra

In this section, we very briefly review vector and matrix operations. Generally, we denote vectors in boldface lowercase, scalars in lowercase italic Roman, and matrices in uppercase Roman.

3.2.1 Vectors

Vectors are always considered to be column vectors. If we need to write one horizontally for the purpose of saving space in a document, we use transpose notation. For example, we denote a vector that consists of three scalar elements as:

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

The Inner Product

The *inner product* of two vectors is a scalar, $c = \mathbf{x}^{\mathrm{T}}\mathbf{y}$. Its value is the sum of products of the corresponding elements of the two vectors:

$$\mathbf{x}^{\mathrm{T}}\mathbf{y} = \sum_i x_i y_i.$$

You will also sometimes see the notation $< \mathbf{x}, \mathbf{y} >$ used for inner product. We do not like this because it looks like an expected value of a random variable. One sometimes also sees the "dot product" notation $\mathbf{x} \cdot \mathbf{y}$ for inner product.

The magnitude of a vector is $|\mathbf{x}| = \sqrt{\mathbf{x}^{\mathrm{T}}\mathbf{x}}$. If $|\mathbf{x}| = 1$, $\mathbf{x}$ is said to be a *unit vector*. If $\mathbf{x}^{\mathrm{T}}\mathbf{y} = 0$, then $\mathbf{x}$ and $\mathbf{y}$ are *orthogonal*. If $\mathbf{x}$ and $\mathbf{y}$ are *orthogonal* unit vectors, they are *orthonormal*.

The Outer Product

While the inner product of two vectors generates a scalar, the outer product generates a matrix.

The *outer product* of two vectors is a matrix, $\mathbf{x} \otimes \mathbf{y} = \mathbf{x}\mathbf{y}^{\mathrm{T}}$. Each element of this matrix can be written as $(\mathbf{x}\mathbf{y}^{\mathrm{T}})_{ij} = x_i y_j$. For example, suppose $\mathbf{x}$ is of 2 dimensions and $\mathbf{y}$ is of 3 dimensions, then the outer product between these two vectors is a matrix of dimension 2×3,

$$\mathbf{x} \otimes \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} = \begin{bmatrix} x_1 y_1 & x_1 y_2 & x_1 y_3 \\ x_2 y_1 & x_2 y_2 & x_2 y_3 \end{bmatrix}.$$

The outer product is also sometimes called the *tensor product*.

The Cross Product

There is yet another product operation between vectors, the *cross product*, which returns a vector. This again contrasts the inner product that returns a scalar and the outer product that returns a matrix.

Given a vector $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^{\mathrm{T}}$, we define the matrix

$$[\mathbf{x}]_\times = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}. \tag{3.1}$$

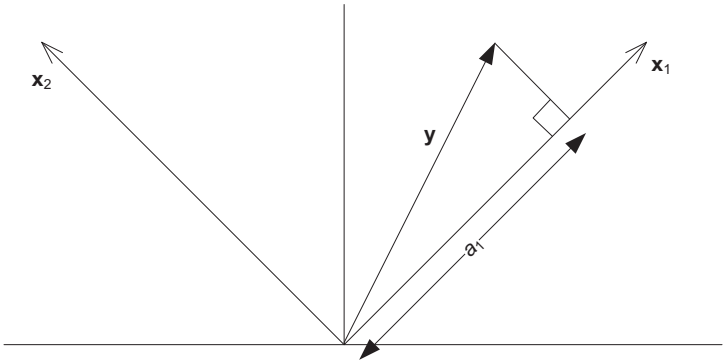


Figure 3.1 $\mathbf{x}_1$ and $\mathbf{x}_2$ are orthonormal bases. The projection of $\mathbf{y}$ onto $\mathbf{x}_1$ has length a_1.

This matrix is *skew-symmetric*. That is, it is equal to the negative of its transpose. The cross product between two vectors $\mathbf{x}$ and $\mathbf{y}$ can then be defined as

$$\mathbf{x} \times \mathbf{y} = [\mathbf{x}]_\times \mathbf{y} = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_2 y_3 - x_3 y_2 \\ x_3 y_1 - x_1 y_3 \\ x_1 y_2 - x_2 y_1 \end{bmatrix}. \tag{3.2}$$

Note that if $\mathbf{x} = \mathbf{y}$, their cross product is the zero vector:

$$\mathbf{x} \times \mathbf{y} = \mathbf{0}. \tag{3.3}$$

The advantage of describing this operation as a matrix is that you can perform the cross product operation (which is only defined for vectors) using matrix multiplication. Thus $\mathbf{x} \times M = [\mathbf{x}]_\times M$ takes the cross product of $\mathbf{x}$ with each column of M.

Linear Independence

Suppose we have n vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$; if we can write $\mathbf{v} = a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots + a_n\mathbf{x}_n$, where $a_1, a_2, \ldots, a_n \in \Re$, then $\mathbf{v}$ is said to be a *linear combination* of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$.

A set of vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ is said to be *linearly independent* if it is impossible to write any of the vectors as a linear combination of the others.

Given d linearly independent vectors, of d dimensions, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d$ defined on $\Re^d$, then any vector $\mathbf{y}$ in the space may be written as $\mathbf{y} = a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots + a_d\mathbf{x}_d$.

Since any d-dimensional real-valued vector $\mathbf{y}$ may be written as a linear combination of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d$, the set $\{\mathbf{x}_i, i = 1, \ldots, d\}$ is called a *basis* set and the vectors are said to *span the space*, $\Re^d$. Any linearly independent set of vectors can be used as a basis (necessary and sufficient). It is often particularly convenient to choose basis sets which are orthonormal.

For example, the following two vectors form a basis for $\Re^2$:

$$\mathbf{x}_1 = \begin{bmatrix} 0 & 1 \end{bmatrix}^\mathrm{T} \qquad \text{and} \qquad \mathbf{x}_2 = \begin{bmatrix} 1 & 0 \end{bmatrix}^\mathrm{T}.$$

This is the familiar Cartesian coordinate system. Here's another basis set[1] for $\Re^2$:

$$\mathbf{x}_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}^\mathrm{T} \qquad \text{and} \qquad \mathbf{x}_2 = \begin{bmatrix} -1 & 1 \end{bmatrix}^\mathrm{T}.$$

[1] Is this set orthonormal?

Table 3.1 Two operations on a set of four symbols

$*$	a	b	c	d
a	a	a	a	a
b	a	b	c	d
c	a	c	c	a
d	a	d	a	d

$+$	a	b	c	d
a	a	b	c	d
b	b	b	b	b
c	c	b	c	b
d	d	b	b	d

If $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d$ span $\Re^d$, and $\mathbf{y} = a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots + a_d\mathbf{x}_d$, then the *components* of $\mathbf{y}$ may be found by

$$a_i = \mathbf{y}^{\mathrm{T}}\mathbf{x}_i \tag{3.4}$$

and a_i is said to be the *projection* of $\mathbf{y}$ onto $\mathbf{x}_i$. In a simple Cartesian geometric interpretation, the inner product of Eq. 3.4 is literally a projection as illustrated in Figure 3.1. However, whenever Eq. 3.4 is used, the term "projection" may be used as well, even in a more general sense (e.g., the coefficients of a Fourier series).

3.2.2 Vector Spaces

A vector space is a set of vectors and two operations, $*$ and $+$, which are defined on those vectors. Furthermore for a set to be a vector space, it must satisfy the following two conditions:

- If $\mathbf{v}_1$ and $\mathbf{v}_2$ are elements of a vector space named, say, V, then $\mathbf{v}_1 + \mathbf{v}_2$ is also in V.
- If $\mathbf{v}_1$ is an element of a vector space named, say, V, and α is a scalar defined consistent with V, then $\alpha\mathbf{v}_1 \in V$.

The property mentioned above has a name, *closure*. We say a vector space is *closed* under vector addition and scalar multiplication.

The only vector spaces that usually concern us are those in which the vectors are composed of real numbers, using the familiar addition and multiplication of real numbers. However, so you can understand these concepts a bit more clearly, let's construct a finite vector space.

An Example Vector Space

Define a set of symbols $S = \{a, b, c, d\}$ and two operations on those symbols named $*$ and $+$. Since we only have four symbols, we get to define the operations with tables as illustrated in Table 3.1.

Define a set V consisting of all the vectors of length 2, using the scalars in S. Let's see if the combination of V and the two operations defined above could be a vector space:

Here are two vectors from V, let's add them:

$$\begin{bmatrix} a \\ b \end{bmatrix} + \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} c \\ b \end{bmatrix}. \tag{3.5}$$

Observe that adding these two vectors produces a vector that is in the same vector space. So at least from this one test, we might have a vector space. To really be sure, we would have to evaluate all the conditions listed above.

3.2.3 Null Spaces

The null space of a matrix, A, is the set of all vectors, $\mathbf{x}$ which satisfy

$$A\mathbf{x} = \mathbf{0}, \tag{3.6}$$

where $\mathbf{0}$ is the vector of all zeros. This is accomplished by first doing row reduction. We provide an example here, to remind the student of how to do row reduction:

Let

$$A = \begin{bmatrix} 4 & 5 & 6 \\ 1 & 2 & 3 \\ 2 & 4 & 6 \end{bmatrix}. \tag{3.7}$$

The first step is to divide the first row by the 1,1 element to make that element equal to one. Recall that if the 1,1 element is zero, you need to swap rows.

$$A = \begin{bmatrix} 1 & 5/4 & 3/2 \\ 1 & 2 & 3 \\ 2 & 4 & 6 \end{bmatrix}.$$

Now multiply the first row by -1 times the 2,1 element, and add it to the second row, replacing the second row. Also perform the same operation on the third row, which will make the 3,1 element equal to zero.

$$A = \begin{bmatrix} 1 & 5/4 & 3/2 \\ 0 & 3/4 & 3/2 \\ 0 & 3/2 & 3 \end{bmatrix}$$

which simplifies to

$$A = \begin{bmatrix} 1 & 5/4 & 3/2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix}.$$

Now, we attempt to set the 3,2 element to zero by multiplying the second row by -1 and adding it to the third row. When we do that, the third row becomes all zeros, and we conclude that this matrix has rank equal to 2. (In truth, the student probably noticed much earlier that the second and third rows were proportional, right?)

$$A = \begin{bmatrix} 1 & 5/4 & 3/2 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix}.$$

We can find the null space by multiplying A by the unknown vector $\mathbf{x} = [x_1 \;\; x_2 \;\; x_3]^{\mathrm{T}}$ and setting the result to zero.

$$\begin{bmatrix} 1 & 5/4 & 3/2 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Setting the product to zero yields the following equations

$$x_1 + \frac{5}{4}x_2 + \frac{3}{2}x_3 = 0 \tag{3.8}$$

$$x_2 + 2x_3 = 0. \tag{3.9}$$

That simplifies to

$$x_1 = x_3 \tag{3.10}$$

$$x_2 = -2x_3. \tag{3.11}$$

Let $x_3 = 1$, and the null vector is $[1 \;\; -2 \;\; 1]^{\mathrm{T}}$. Since the rank is 2, the null space consists of this one vector and all scalar multiples of it. Division of $[1 \;\; -2 \;\; 1]^{\mathrm{T}}$ by its magnitude, $\sqrt{6}$, produces a unit vector, $[0.4082 \; -0.8165 \; 0.4082]^{\mathrm{T}}$.

Right and Left Null Space

Above, we solved the problem

$$A\mathbf{x} = \mathbf{0}.$$

Instead, however, we might have been confronted with the following problem

$$\mathbf{x}A = \mathbf{0},$$

which is referred to as finding the *left null space*. The left null space of a matrix, A is simply the null space of A^{T}.

3.2.4 Function Spaces

Think about a function, say $f(x) = x^2$, defined on the integers greater than zero and less than six. You could list the values of $f(x)$ as an ordered set: $[1, 4, 9, 16, 25]$, which looks exactly like a vector. So it is perfectly fine to think of a function as a vector. Of course, there is one slight problem. If we define our function on the real numbers, we end up with a vector with infinite dimensionality. But, except for the fact that we cannot list all the elements, infinite dimensionality is not a problem.

The concept of orthogonality can easily be extended to continuous functions by simply thinking of a function as an infinite-dimensional vector. Just list all the values of $f(x)$ as x varies between, say, a and b. If x is continuous, then there are an infinite number of possible values of x between a and b. But that should not stop us – we cannot enumerate them, but we can still think of a vector containing all the values of $f(x)$. Now, the concept of summation that we defined for finite-dimensional vectors turns into integration, and an inner product may be written

$$< f(x), g(x) >= \int_a^b f(x)g(x)dx. \tag{3.12}$$

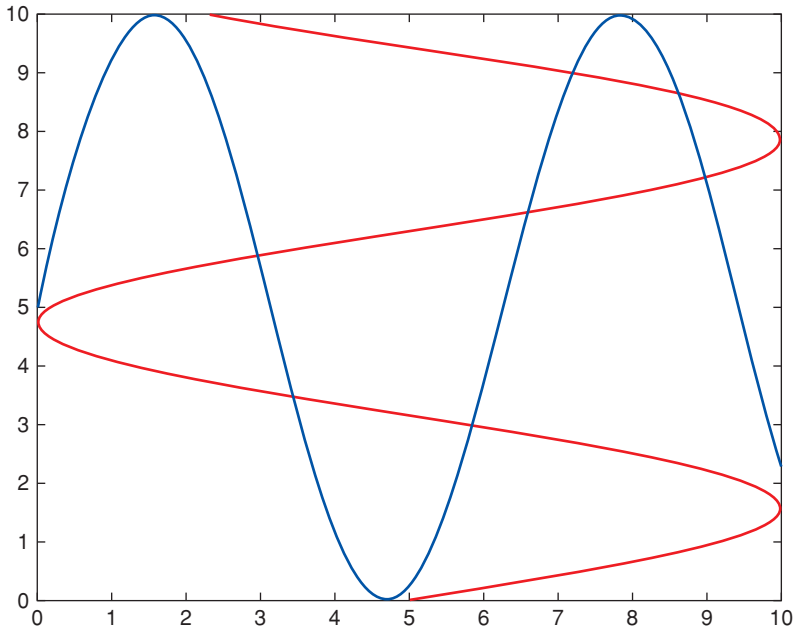


Figure 3.2 The mental image that appears in the mind of someone who confuses *orthogonal* and *perpendicular* in misunderstanding the statement "The cosine is orthogonal to the sine."

The concepts of orthogonality and orthonormality hold for this definition of the inner product as well. If the integral is equal to zero, we say the two functions are orthogonal. So the transition from orthogonal vectors to orthogonal functions is not that difficult. With an infinite number of dimensions, it is impossible to visualize orthogonal as "perpendicular," of course, so you need to give up on thinking about things being perpendicular. Just recall the definition and use it.

However, in case you like silly misunderstandings, Figure 3.2 illustrates an incorrect interpretation of the statement "The sine and the cosine are orthogonal."[2]

3.2.5 Linear Transformations

A *linear transformation*, A, is simply a matrix. Suppose A is $m \times d$. If applied to a vector $\mathbf{x} \in \Re^d$, $\mathbf{y} = A\mathbf{x}$, then $\mathbf{y} \in \Re^m$. So A took a vector from one vector space, $\Re^d$, and produced a vector in $\Re^m$. If that vector $\mathbf{y}$ could have been produced by applying A to one and only one vector in $\Re^d$, then A is said to be *one-to-one*. Now suppose that there are no vectors in $\Re^m$ that cannot be produced by applying A to some vector in $\Re^d$; in that case, A is said to be *onto*. If A is one-to-one and onto, then A^{-1} exists. Two matrices A and B are *conformable* if the matrix multiplication $C = AB$ makes sense.

[2] Can you prove this statement?

Some important (and often forgotten) properties[3]: If A and B are conformable, then

$$(AB)^{\mathrm{T}} = B^{\mathrm{T}}A^{\mathrm{T}} \tag{3.13}$$

and

$$(AB)^{-1} = B^{-1}A^{-1} \tag{3.14}$$

if A and B are invertible at all.

A couple of other useful properties related to determinant (det) and trace (tr) are

$$\det(AB) = \det(BA) \qquad \text{and} \qquad \mathrm{tr}(AB) = \mathrm{tr}(BA)$$

which is only true, of course, if A and B are square.

Orthonormal Transformation

If a matrix A satisfies

$$AA^{\mathrm{T}} = A^{\mathrm{T}}A = I \tag{3.15}$$

then obviously, the transpose of the matrix is the inverse as well, and A is said to be an *orthonormal transformation* (OT), which will correspond geometrically to a rotation. If A is a $d \times d$ orthonormal transformation, then the columns of A are orthonormal, linearly independent, and form a basis spanning the space of $\Re^d$. For $\Re^3$, three convenient OTs are the rotations about the Cartesian axes (assuming the right-handed coordinate system[4]) with θ being the rotation angle defined in the counterclockwise direction.

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}, R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

Suppose R is an OT, and $\mathbf{y} = R\mathbf{x}$, then both $\mathbf{x}$ and $\mathbf{y}$ have the same magnitude:

$$|\mathbf{y}| = |\mathbf{x}| \tag{3.17}$$

Although the three rotation matrices in Eq. 3.16 share some common characteristics, i.e., there are always a column of zeros and a single 1 as well as a row of zeros and a single 1, these characteristics are not the necessary condition to make a rotation matrix. In fact, rotation matrices share two interesting properties:

- the product of two rotation matrices (i.e., OT) is a rotation matrix, and
- any OT can be considered a rotation matrix, where the rotation is about some axis.

In the following, we show an example that well demonstrates the above two properties.

[3] We assume you know the meanings of transpose, inverse, determinant, and trace. If you do not, you should not be taking this course.

[4] Right-handed coordinate system indicates that the axis about which the rotation (indicated by circling the fingers) occurs points toward the observer.

Suppose two rotations are performed in sequence with the first one rotating about x by 20 degrees, followed by a second one rotating about z by 45 degrees, both in the counter-clockwise direction, then the composite transformation matrix, R_{zx} can be calculated by $R_{zx} = R_z R_x$, where

$$R_x = \begin{bmatrix} 1.0000 & 0 & 0 \\ 0 & 0.9397 & 0.3420 \\ 0 & -0.3420 & 0.9397 \end{bmatrix}, R_z = \begin{bmatrix} 0.7071 & -0.7071 & 0 \\ 0.7071 & 0.7071 & 0 \\ 0 & 0 & 1.0000 \end{bmatrix},$$

and

$$R_{zx} = R_z R_x = \begin{bmatrix} 0.7071 & -0.7071 & 0 \\ 0.6645 & 0.6645 & 0.3420 \\ -0.2418 & -0.2418 & 0.9397 \end{bmatrix}.$$

As we can see, in the composite matrix R_{zx}, the columns are orthogonal, yet we do not see the column of zeros and a single 1 that characterizes the three rotation matrices around x, y, and z in Eq. 3.16.

Positive Definite

A matrix A is *positive definite* if

$$y = \mathbf{x}^{\mathrm{T}} A \mathbf{x} > 0, \forall \mathbf{x} \in \Re^d, \mathbf{x} \neq 0$$

A matrix A is *positive semidefinite* if

$$y = \mathbf{x}^{\mathrm{T}} A \mathbf{x} \geq 0, \forall \mathbf{x} \in \Re^d, \mathbf{x} \neq 0$$

$\mathbf{x}^{\mathrm{T}} A \mathbf{x}$ is called a *quadratic form*. Since the quadratic form always returns a scalar, a positive definite or a positive semidefinite matrix is always square.

3.2.6 Derivatives and Derivative Operators

The derivative of a quadratic form is particularly useful[5]:

$$\frac{d}{dx}(\mathbf{x}^{\mathrm{T}} A \mathbf{x}) = (A + A^{\mathrm{T}})\mathbf{x}.$$

Since we mentioned derivatives, we will mention a couple of other vector calculus things:

Suppose $f(\mathbf{x})$ is a scalar function of $\mathbf{x}$, $\mathbf{x} \in \Re^d$, then

$$\frac{df}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_d} \end{bmatrix}^{\mathrm{T}}, \tag{3.18}$$

and is called the *gradient*. This will be often used when we talk about edges in images, and $f(\mathbf{x})$ will be the brightness as a function of the two spatial directions. If $\mathbf{f}(\mathbf{x})$ is vector-valued, then the derivative is a matrix

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_d} & \frac{\partial f_2}{\partial x_d} & \cdots & \frac{\partial f_m}{\partial x_d} \end{bmatrix}, \tag{3.19}$$

and is called the *Jacobian*.

[5] What happens here if A is symmetric?

One more: if $f(\mathbf{x})$ is scalar-valued, the matrix of second derivatives

$$\frac{d^2 f}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix}, \tag{3.20}$$

is called the *Hessian*.

Here, we introduce a new notation, a vector containing only operators,

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x_1} & \frac{\partial}{\partial x_2} & \cdots & \frac{\partial}{\partial x_d} \end{bmatrix}^{\mathrm{T}}. \tag{3.21}$$

It is important to note that this is an *operator*, not a vector. We will do linear algebra sorts of things with it, but by itself, it has no value, not even really any meaning – it must be applied to something to have any meaning. For most of this book, we will deal with two-dimensional images, and with the two-dimensional form of this operator,

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{bmatrix}^{\mathrm{T}}. \tag{3.22}$$

Apply this operator to a scalar, f, and we get a vector that does have meaning, the gradient of f:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix}^{\mathrm{T}}. \tag{3.23}$$

Similarly, if $\mathbf{f}$ is a vector, we may define the *divergence* using the inner (dot) product (in all the following definitions, only the two-dimensional form of the ∇ operator defined in Eq. 3.21 is used. However, remember that the same concepts apply to operators of arbitrary dimension):

$$div\, \mathbf{f} = \nabla \mathbf{f} = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \frac{\partial f_1}{\partial x} + \frac{\partial f_2}{\partial y}. \tag{3.24}$$

We will also have opportunity to use the outer product of the ∇ operator with a matrix, which is the Jacobian:

$$\nabla \times \mathbf{f} = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \begin{bmatrix} f_1 & f_2 \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_2}{\partial x} \\ \frac{\partial f_1}{\partial y} & \frac{\partial f_2}{\partial y} \end{bmatrix}. \tag{3.25}$$

3.2.7 Eigenvalues and Eigenvectors

If matrix A and vector $\mathbf{x}$ are conformable, then one may write the *characteristic equation*

$$A\mathbf{x} = \lambda \mathbf{x}, \lambda \in \Re. \tag{3.26}$$

Since $A\mathbf{x}$ is a linear operation, A may be considered as a transformation that maps $\mathbf{x}$ onto itself with only a change in length. There may be more than one *eigenvalue*,[6] λ, which satisfies Eq. 3.26. For $\mathbf{x} \in \Re^d$, A will have exactly d eigenvalues (which are not, however, necessarily distinct). These may be found by solving $\det(A - \lambda I) = 0$. (But for $d > 2$, we do not recommend this method. Use a numerical package instead.)

[6] "Eigen-" is the German prefix meaning "principal" or "most important." These are *not* named for Mr. Eigen.

Given some eigenvalue λ, which satisfies Eq. 3.26, the corresponding $\mathbf{x}$ is called the corresponding *eigenvector*.[7]

3.2.8 Eigendecomposition

If a positive semidefinite matrix, A, is also symmetric, then it may be written as $A = BB^{\mathrm{T}}$ for some matrix B. Many of the matrices encountered in Computer Vision are positive semidefinite. In particular, covariance matrices, which we will encounter several times, have this property.

Assuming we know how to compute eigenvalues and eigenvectors, a positive semidefinite matrix can be written as

$$A = E\Lambda E^{\mathrm{T}}, \tag{3.27}$$

where E is a matrix in which each column is an eigenvector of A, and Λ is a square, diagonal matrix with corresponding eigenvalues on the diagonal. Should A not be positive semidefinite and symmetric, but still invertible, the form simply changes to

$$A = E\Lambda E^{-1}, \tag{3.28}$$

We can find the null space easily by taking the eigendecomposition and considering all the eigenvectors that correspond to eigenvalues of zero. As an example, using the same matrix as we used in Eq. 3.7 earlier (which is *not* symmetric), we find

$$A = \begin{bmatrix} 4 & 5 & 6 \\ 1 & 2 & 3 \\ 2 & 4 & 6 \end{bmatrix} = \begin{bmatrix} 0.7560 & 0.9469 & 0.4082 \\ 0.2927 & -0.1438 & -0.8165 \\ 0.5855 & -0.2877 & 0.4082 \end{bmatrix} \times \begin{bmatrix} 10.5826 & 0 & 0 \\ 0 & 1.4174 & 0 \\ 0 & 0 & -0.0000 \end{bmatrix} \begin{bmatrix} 0.3727 & 0.6398 & 0.9069 \\ 0.7585 & -0.0884 & -0.9353 \\ 0.0000 & -0.9798 & 0.4899 \end{bmatrix}.$$

Observe that the right column of E (the left-hand matrix) is the eigenvector corresponding to the zero eigenvalue, and it is precisely the null space we found in the example in section 3.2.3. Also note that because A is not symmetric, we had to use E^{-1} for the third matrix in the product.

In this example, if we had two zero eigenvalues, the two corresponding eigenvectors would be a basis for the space of elements of the null space. That is, any vector that is a linear sum of those two vectors would be in the null space.

But of course, this only works for square matrices!

In the next section, we generalize this concept and develop *singular value decomposition*, or SVD.

[7] For any given matrix, there are only a few eigenvalue/eigenvector pairs.

3.2.9 Singular Value Decomposition

While eigendecomposition is restricted to square matrices, SVD can be applied to any matrix. SVD decomposes a matrix A into the product of three matrices:

$$A = UDV^{\mathrm{T}}, \tag{3.29}$$

where U and V are orthonormal matrices, and D is diagonal with values on the diagonal sorted from largest to smallest.

To understand what this means, suppose you had a machine that could compute a matrix decomposition just as described in Eq. 3.29, and consider using it to decompose A. Now, since $A = UDV^{\mathrm{T}}$, we know that $A^{\mathrm{T}} = VDU^{\mathrm{T}}$. Multiplying A on the right by A^{T} produces $AA^{\mathrm{T}} = UDV^{\mathrm{T}}VDU^{\mathrm{T}}$, but since V is orthonormal, $V^{\mathrm{T}}V = I$, and AA^{T} may be written as

$$AA^{\mathrm{T}} = UD^2U^{\mathrm{T}}. \tag{3.30}$$

and the columns of U (see Eq. 3.27) will be the eigenvectors of AA^{T}. Similarly, one can show that the columns of V will be the eigenvectors of $A^{\mathrm{T}}A$. D will be a diagonal matrix with the square root of the eigenvalues of $A^{\mathrm{T}}A$ on the diagonal sorted in descending order.[8]

The diagonal elements of D are referred to as the *singular values* of A, so the singular values of A are the square roots of the eigenvalues of AA^{T}.

This observation tells us one way to compute the SVD: use eigendecomposition. But more important: eigendecomposition can only be applied to square matrices, and both AA^{T} and $A^{\mathrm{T}}A$ are square, for ANY matrix A.

Now suppose our problem is to find the null space of a 3×3 matrix that is of rank 2, just like the problem in section 3.2.3. Using the same original matrix, we compute

$$AA^{\mathrm{T}} = \begin{bmatrix} 77 & 32 & 64 \\ 32 & 14 & 28 \\ 64 & 28 & 56 \end{bmatrix}, \; A^{\mathrm{T}}A = \begin{bmatrix} 21 & 30 & 39 \\ 30 & 45 & 60 \\ 39 & 60 & 81 \end{bmatrix}. \tag{3.31}$$

Decomposing AA^{T} using eigendecomposition, we find

$$\begin{aligned} AA^{\mathrm{T}} = &\begin{bmatrix} 0.7242 & -0.6896 & -0.0000 \\ 0.3084 & 0.3239 & 0.8944 \\ 0.6168 & 0.6477 & -0.4472 \end{bmatrix} \begin{bmatrix} 145.1397 & 0 & 0 \\ 0 & 1.8603 & 0 \\ 0 & 0 & -0.0000 \end{bmatrix} \\ &\times \begin{bmatrix} 0.7242 & 0.3084 & 0.6168 \\ -0.6896 & 0.3239 & 0.6477 \\ -0.0000 & 0.8944 & -0.4472 \end{bmatrix}. \end{aligned} \tag{3.32}$$

[8] All numerical packages that compute SVD do this type of sorting

Similarly, the eigendecomposition of $A^{\mathrm{T}}A$ is

$$A^{\mathrm{T}}A = \begin{bmatrix} 0.3684 & -0.8352 & -0.4082 \\ 0.5565 & -0.1536 & 0.8165 \\ 0.7447 & 0.5280 & -0.4082 \end{bmatrix} \begin{bmatrix} 145.1397 & 0 & 0 \\ 0 & 1.8603 & 0 \\ 0 & 0 & -0.0000 \end{bmatrix} \times \begin{bmatrix} 0.3684 & 0.5565 & 0.7447 \\ -0.8352 & -0.1536 & 0.5280 \\ -0.4082 & 0.8165 & -0.4082 \end{bmatrix}. \tag{3.33}$$

Choosing U from the eigenvectors of AA^{T} and V from the eigenvectors of $A^{\mathrm{T}}A$, and taking the square roots of the diagonal elements of the center matrix of either one, we can write

$$U\Lambda V^{\mathrm{T}} = \begin{bmatrix} 0.7242 & -0.6896 & -0.0000 \\ 0.3084 & 0.3239 & 0.8944 \\ 0.6168 & 0.6477 & -0.4472 \end{bmatrix} \begin{bmatrix} 12.0474 & 0 & 0 \\ 0 & 1.3639 & 0 \\ 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0.3684 & 0.5565 & 0.7447 \\ -0.8352 & -0.1536 & 0.5280 \\ -0.4082 & 0.8165 & -0.4082 \end{bmatrix}, \tag{3.34}$$

which is the SVD of A. Thinking about the null space of A, we realize that the null vector (in this particular case) of A is the column of V (or row of V^{T}) corresponding to the zero singular value.

3.3 Introduction to Function Minimization

Minimization[9] of functions is a pervasive element of engineering: One is always trying to find the set of parameters that minimizes some function of those parameters. Notationally, we state the problem as: Find the vector $\hat{\mathbf{x}}$ that produces a minimum of some function $H(\mathbf{x})$:

$$\hat{H} = \min_{\hat{\mathbf{x}}} H(\mathbf{x}) \tag{3.35}$$

where $\mathbf{x}$ is some d-dimensional parameter vector, and H is a scalar function of $\mathbf{x}$, often referred to as an *objective function*. We denote the $\mathbf{x}$ that results in the minimal H as $\hat{\mathbf{x}}$

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} H(\mathbf{x}). \tag{3.36}$$

There are two distinct reasons for learning how to minimize a function:

- We may need to find a way to perform some process that has a minimal cost, minimal run time, minimal programming difficulty, etc.
- We may need to solve some problem that we don't (immediately) know how to solve. For example, we might seek to draw a contour in an image that separates dark areas from light areas with the additional desire that the regions should be as smooth as possible.

[9] In this book, essentially *every* computer vision topic will be discussed in terms of some sort of minimization, so get used to it!

In the second case, our approach might be to figure out some scalar function that will be minimal if the conditions are what we are seeking. For example, suppose we chose an objective function something like[10]

$$H(C) = \int_{\text{inside(C)}} |f(x,y) - \hat{f}| dxdy + \int_{\text{outside(C)}} |f(x,y) - \check{f}| dxdy \tag{3.37}$$

where C is the (unknown) contour, $\hat{f}$ is the average brightness inside the contour, and $\check{f}$ is the average brightness outside the contour. Finding the C that minimizes this objective function would certainly divide the image into the inside and outside nicely. Of course, we don't know C, $\hat{f}$, or $\check{f}$. So we just created a new problem: given the objective function, how do we minimize it?

In the next subsection we illustrate some simple, general methods to begin to think about function minimization. The topic of function minimization will come up again in section 6.5.1, where minimization will be used to smooth an image while preserving sharp edges; in section 5.4.2, where minimization will find good ways to detect edges; and in section 10.3.2, where it will be used to fit lines, just to mention a few.

3.3.1 Gradient Descent

The most straightforward way to minimize a function[11] is to set its derivative to zero:

$$\nabla H(\mathbf{x}) = 0, \tag{3.38}$$

where ∇ is the gradient operator. Eq. 3.38 results in a set of equations, one for each element of $\mathbf{x}$, which must be solved simultaneously.

$$\begin{cases} \frac{\partial}{\partial x_1} H(\mathbf{x}) = 0 \\ \frac{\partial}{\partial x_2} H(\mathbf{x}) = 0 \\ \quad \cdots \\ \frac{\partial}{\partial x_d} H(\mathbf{x}) = 0. \end{cases} \tag{3.39}$$

Such an approach is practical only if the system of Eq. 3.39 is solvable. This may be true if $d = 1$, or if H is at most quadratic in $\mathbf{x}$.

Exercise 3.1: Find the vector $\mathbf{x} = [x_1, x_2, x_3]^{\mathrm{T}}$ that minimizes

$$H = ax_1^2 + bx_1 + cx_2^2 + dx_3^2$$

where a, b, c, and d, are known constants.

[10] An equation similar to this will occur when we discuss segmentation.

[11] The authors get *very* annoyed at improper use of the word "optimal." If you didn't solve a formal optimization problem to get your result, you didn't come up with the "optimal" anything.

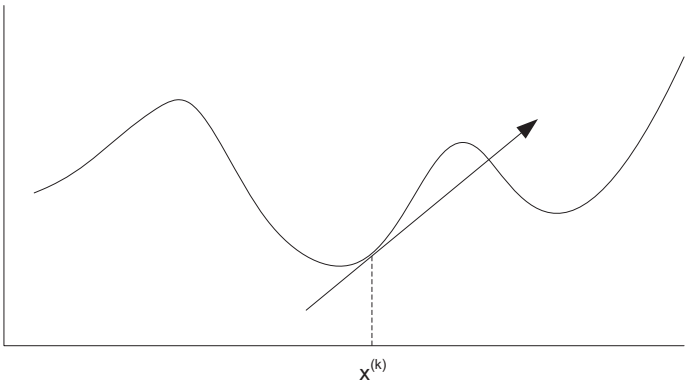


Figure 3.3 The sign of derivative is always away from the minimum.

Solution

$$\begin{cases} \frac{\partial H}{\partial x_1} = 2ax_1 + b = 0 \\ \frac{\partial H}{\partial x_2} = 2cx_2 = 0 \\ \frac{\partial H}{\partial x_3} = 2dx_3 = 0 \end{cases}$$

is minimized by

$$x_3 = x_2 = 0, x_1 = \frac{-b}{2a}.$$

If H is some function of order higher than two, or is transcendental, the technique of setting the derivative equal to zero may result in a set of simultaneous equations that are (usually) impractical to solve algebraically, and we must resort to numerical techniques. The first of these is *gradient descent*.

In one dimension, the utility of the gradient is easy to see. At a point $x^{(k)}$ where k is the iteration index (Figure 3.3), the derivative points ***away from*** the minimum. That is, in one dimension, its sign will be positive on an "uphill" slope.

Thus, in pursuit of a minimum, to find a new point, $x^{(k+1)}$, we let

$$x^{(k+1)} = x^{(k)} - \alpha \left.\frac{\partial H}{\partial x}\right|_{x^{(k)}} \tag{3.40}$$

where α is some "small" constant.

In a problem where $\mathbf{x}$ is a vector with d variables, we write

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \left.\nabla H(\mathbf{x})\right|_{\mathbf{x}^{(k)}}. \tag{3.41}$$

It is not immediately obvious in Eq. 3.41 how to choose the variable α that is also referred to as the *learning rate*. If α is too small, the iteration of Eq. 3.41 will take too long to converge. If α is too large, the algorithm may become unstable and never find the minimum.

We can find an estimate for α by considering the well-known Newton-Raphson method for finding roots: In one dimension, we expand the function $H(x)$ in a Taylor series about the point $x^{(k)}$ and truncate, assuming all higher order terms are zero,

$$H\left(x^{(k+1)}\right) = H\left(x^{(k)}\right) + \left(x^{(k+1)} - x^{(k)}\right)H'\left(x^{(k)}\right).$$

Since we want $x^{(k+1)}$ to be a zero of H, we set

$$H\left(x^{(k)}\right) + \left(x^{(k+1)} - x^{(k)}\right)H'\left(x^{(k)}\right) = 0, \tag{3.42}$$

and find that to estimate a root, we should use

$$x^{(k+1)} = x^{(k)} - \frac{H\left(x^{(k)}\right)}{H'\left(x^{(k)}\right)}. \tag{3.43}$$

In optimization, however, we are not finding roots, but rather minimizing a function, so how does knowing how to find roots help us? The minima of the function are the roots of its derivative, and the "gradient descent" algorithm becomes

$$x^{(k+1)} = x^{(k)} - \frac{H'\left(x^{(k)}\right)}{H''\left(x^{(k)}\right)}. \tag{3.44}$$

In higher dimensions, Eq. 3.44 becomes

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{H}^{-1}\nabla H, \tag{3.45}$$

where $\mathbf{H}$ is the Hessian matrix of second derivatives, which we mentioned earlier in this chapter:

$$\mathbf{H} = \left[\frac{\partial^2}{\partial x_i \partial x_j} H(\mathbf{x})\right], \tag{3.46}$$

and comparing Eq. 3.41 with Eq. 3.45, we have

$$\alpha = \mathbf{H}^{-1}.$$

Note that α is not a scalar any more.

Exercise 3.2: Given a set of x, y data pairs $\{(x_i, y_i)\}$ and a function of the form

$$y = ae^{bx}, \tag{3.47}$$

find the parameters a and b that minimize

$$H(a, b) = \sum_i \left(y_i - ae^{bx_i}\right)^2. \tag{3.48}$$

Solution

We can solve this problem with the linear approach by observing that $\ln y = \ln a + bx$ and redefining variables $g = \ln y$ and $r = \ln a$. With these substitutions, Eq. 3.48 becomes

$$H(r, b) = \sum_i (g_i - r - bx_i)^2 \tag{3.49}$$

$$\frac{\partial H}{\partial b} = 2\sum_i (g_i - r - bx_i)(-x_i) \tag{3.50}$$

$$\frac{\partial H}{\partial r} = 2\sum_i (g_i - r - bx_i)(-1). \tag{3.51}$$

We can solve this minimization problem by either setting the derivatives equal to zero and finding the root of the simultaneous equations, or using the gradient descent algorithm if

the equations are not solvable. Assume the former is adopted, setting Eq. 3.50 to zero, we have

$$\sum_i g_i x_i - \sum_i r x_i - \sum_i b x_i^2 = 0 \tag{3.52}$$

or

$$r \sum_i x_i + b \sum_i x_i^2 = \sum_i g_i x_i \tag{3.53}$$

and from Eq. 3.51

$$\sum_i g_i - r \sum_i 1 - b \sum_i x_i = 0 \tag{3.54}$$

or

$$N r + b \sum_i x_i = \sum_i g_i \tag{3.55}$$

where N is the number of data points. Eqs. 3.53 and 3.55 are two simultaneous linear equations in two unknowns that are readily solved.

If the gradient descent algorithm is adopted, then upon selecting an initial value of $[\,r \quad b\,]^{\mathrm{T}}$, the parameters can be updated using

$$\begin{bmatrix} r \\ b \end{bmatrix}^{(k+1)} = \begin{bmatrix} r \\ b \end{bmatrix}^{(k)} - \alpha \begin{bmatrix} \frac{\partial H}{\partial r} \\ \frac{\partial H}{\partial b} \end{bmatrix}.$$

See [3.1, 3.2, 3.3] , for more sophisticated descent techniques, such as the conjugate gradient method.

3.3.2 Local vs. Global Minima

Gradient descent suffers from a serious problem: its solution is strongly dependent on the starting point. If started in a "valley," it will find the bottom of *that* valley. We have no assurance that this particular minimum is the lowest, or "global," minimum.

Before continuing, we will find it useful to distinguish two kinds of nonlinear optimization problems:

- **Combinatorial optimization**. In this case, the variables have discrete values, typically 0 and 1. With $\mathbf{x}$ consisting of d binary-valued variables, 2^d possible values exist for $\mathbf{x}$. Minimization of $H(\mathbf{x})$ then (in principle) consists of simply generating each possible value for $\mathbf{x}$ and consequently of $H(\mathbf{x})$, and choosing the minimum. Such "exhaustive search" is in general not practical due to the exponential explosion of possible values. We will find that simulated annealing provides an excellent approach to solving combinatorial optimization problems.
- **Image optimization**. Images have a particular property: Each pixel is influenced only by its local neighborhood (this will be explained in more detail later); however, the pixel values are continuously valued, and there are typically many thousand such variables. We will find that mean field annealing is appropriate for the solution of these problems. See more discussion of mean field annealing in section 6.5.3.

3.3.3 Simulated Annealing

We will base much of the following discussion of minimization techniques on an algorithm known as "simulated annealing" (SA), which proceeds below as Algorithm 3.1. (See the book by Aarts and van Laarhoven for more detail [3.4].)

Algorithm 3.1: Simulated annealing.

Data: Choose (at random) an initial value of $\mathbf{x}$, and an initial value of $T > 0$.
Result: $\hat{\mathbf{x}}$ that minimizes H.

```
1  while T > T_min do
2      Generate a point y which is a "neighbor" of x. (The exact definition of "neighbor"
       will be discussed soon.);
3      if H(y) < H(x) then
4          Replace x with y;
5      end also denoted as "accepting y"
6      else
7          Compute P_y = exp(-(H(y)-H(x))/T);
8          if P_y ≥ R then
9              Replace x with y, where R is a random number uniformly distributed
               between 0 and 1;
10         end
11     end
12     Decrease T slightly;
13 end
```

Simulated annealing is most easily understood in the context of combinatorial optimization. In this case, one definition of a "neighbor" of a vector $\mathbf{x}$ is another vector $\mathbf{y}$, such that only one of the elements of $\mathbf{x}$ is changed (discretely) to create $\mathbf{y}$.[12] Thus, if $\mathbf{x}$ is binary and of dimension d, one may choose a neighboring $\mathbf{y} = \mathbf{x} \oplus \mathbf{z}$, where $\mathbf{z}$ is a binary vector in which exactly one element is non-zero, and that element is chosen at random, and $\oplus$ represents exclusive OR.

In lines 3 through 5 of the algorithm, we perform a descent. Thus we "always fall down hill."

In lines 7 through 10, we provide a mechanism for sometimes making uphill moves. Initially, we ignore the parameter T and note that if $\mathbf{y}$ represents an uphill move, the probability of accepting $\mathbf{y}$ is proportional to $\exp(-(H(\mathbf{y}) - H(\mathbf{x})))$. Thus, uphill moves can occur, but are exponentially less likely to occur as the size of the uphill move becomes larger. The likelihood of an uphill move is, however, strongly influenced by T. Consider the case that T is very large. Then $\frac{H(\mathbf{y})-H(\mathbf{x})}{T} \ll 1$ and $P_{\mathbf{y}} \approx 1$. Thus, all moves will be accepted. As T is gradually reduced, uphill moves become gradually less likely until for low values of T ($T \ll (H(\mathbf{y}) - H(\mathbf{x}))$, such moves are essentially impossible.

[12] Thus the set of neighbors of $\mathbf{x}$ consists of all $\mathbf{x}$'s of Hamming distance $= 1$.

sum		number of ways
0		0
1		1
2	1-1	1
3	2-1,1-2	2
4	1-3,3-1,2,2	3
5	2-3,3-2,4-1,1-4	4
6	1-5,5-1,2-4,4-2,3-3	5
7	3-4,4-3,2-5,5-2,6-1,1-6	6
8	2-6,6-2,3-5,5-3,4-4	5
9	3-6,6-3,4-5,5-4	4
10	4-6,6-4,5-5	3
11	5-6,6-5	2
12	6-6	1

Figure 3.4 The possible ways to roll two dice.

One may consider an analogy to physical processes in which the state of each variable (one or zero) is analogous to the spin of a particle (up or down). At high temperatures, particles randomly change state, and if temperature is gradually reduced, minimum energy states are achieved. The parameter T in Algorithm 3.1 is thus analogous to (and often referred to as) temperature, and this minimization technique is therefore called "simulated annealing."

3.4 A Brief Review of Probability

Let us imagine a statistical experiment: rolling two dice. It is possible to roll any number between two and twelve (inclusive), but as we know, some numbers are more likely than others. To see this, consider the possible ways to roll a five.

We see from Figure 3.4 that there are four possible ways to roll a five with two dice. Each event is **independent**. That is, the chance of rolling a two with the second die does not depend at all on what is rolled with die number 1.

Independence of events has an important implication. It means that the *joint probability* of the two events is equal to the product of their individual probabilities, and the conditional probabilities:

$$Pr(a, b) = Pr(a|b)P(b) = Pr(a)Pr(b) = Pr(b|a)Pr(a). \tag{3.56}$$

In Eq. 3.56, the symbols a and b represent **events**, e.g., the rolling of a six. $Pr(b)$ is the probability of such an event occurring, and $Pr(a|b)$ is the **conditional probability** of event a occurring, given that event b has occurred.

In Figure 3.4, we tabulate all the possible ways of rolling two dice, and show the resulting number of different ways that the numbers from 2 to 12 can occur. We note that 6 different events can lead to a 7 being rolled. Since each of these events is equally probable (1 in 36),

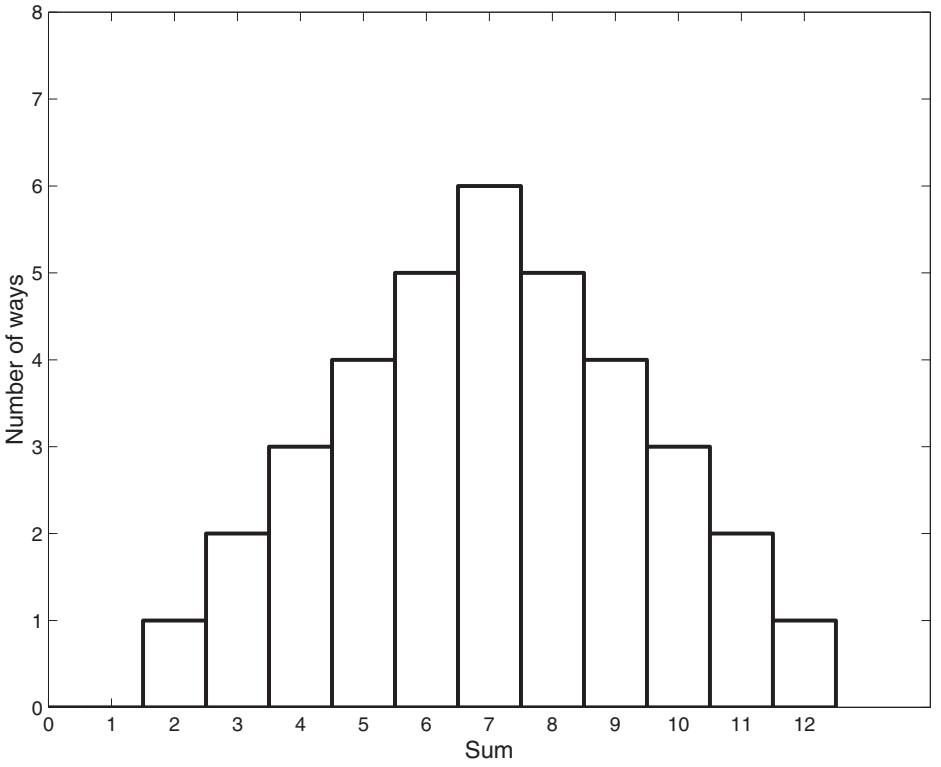


Figure 3.5 The information of Figure 3.4 in graphical form.

then a 7 is the most likely roll of two dice. In Figure 3.5 the information from Figure 3.4 is presented in graphical form.

In Computer Vision, we are most often interested in the probability of a particular measurement occurring. We have a problem, however, when we try to plot a graph such as Figure 3.5 for a continuously valued function. For example, how do we ask the question: "What is the probability that a man is six feet tall?" Clearly, the answer is zero, for an infinite number of possibilities could occur arbitrarily close to 6.0 feet (we might equally well ask, "What is the probability that a man is (exactly) 6.314159267 feet tall?"). Still, we know intuitively that the likelihood of a man being six feet tall is higher than the likelihood of his being ten feet tall. We need some way of quantifying this intuitive notion of likelihood.

One question that does make sense is, "What is the probability that a man is **less than** six feet tall?" Such a function is referred to as a *probability distribution function*

$$P(x) = Pr(h < x) \tag{3.57}$$

for some measurement, h. Figure 3.6 illustrates the probability distribution function for the result of rolling two dice.

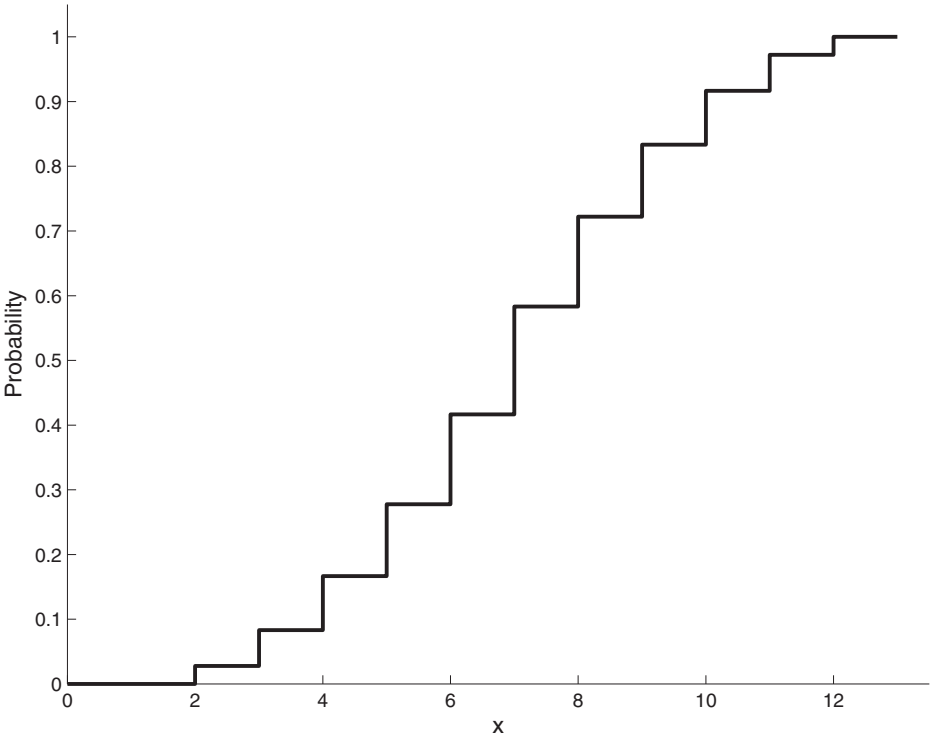


Figure 3.6 The probability distribution of Figure 3.5, showing the probability of rolling two dice to get a number LESS than x. Note that the curve is steeper at the more likely numbers.

Another well-formed question would be, "What is the probability that a man's height is between x and $x + \Delta x$?" Such a question is easily answered in terms of the density function:

$$Pr(x \leq h < x + \Delta x) = Pr(h < x + \Delta x) - Pr(h < x) = P(x + \Delta x) - P(x)$$

Dividing by Δx and taking the limit as $\Delta x \to 0$, we see that we may define the *probability density function* as the derivative of the distribution function:

$$p(x) = \frac{d}{dx}P(x) \tag{3.58}$$

where $p(x)$ has all the properties that we desire. It is well defined for continuously valued measurements and it has a maximum value for those values of the measurement that are intuitively most likely. Furthermore,

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{3.59}$$

which we must require, since **some** value will certainly occur.

3.5 Assignments

Assignment 3.1: Determine whether the functions $\sin x$ and $\sin 2x$ might be orthonormal or orthogonal functions.

Assignment 3.2: Find the OT corresponding to a counterclockwise rotation of 30° about the z axis. Prove the columns of the resulting matrix are a basis for $\mathbb{R}^3$.

Assignment 3.3: Prove Eq. 3.17. Hint: use Eq. 3.13. Eq. 3.15 might be useful as well.

Assignment 3.4: A positive definite matrix has positive eigenvalues. Prove this. (For that matter, is it even true?)

Assignment 3.5: In section 3.2.2, we defined a peculiar set of scalars, called S, and coupled it with two strange operations called $*$ and $+$, and speculated that might be a vector space. One example was given, which showed that it might possibly be. What do you think? Can you find a counterexample?

Assignment 3.6: In section 3.2.2, we defined a peculiar set of scalars, called S, and coupled it with two strange operations called * and +, and speculated that might be a vector space. How would you define orthogonality in this space? Would you have to find something that acts like zero? How would you define an inner product? Discuss.

Assignment 3.7: Does the function $y = xe^{-x}$ have a unique value x that minimizes y? If so, can you find it by taking a derivative and setting it equal to zero? Suppose this problem requires gradient descent to solve. Write the algorithm you would use to find the x that minimizes y.

Assignment 3.8: We need to solve a minimization problem using gradient descent. The function we are minimizing is $\sin x + \ln y$. Which of the following is the expression for the gradient that you need in order to do gradient descent?

(a) $\cos x + 1/y$ (b) $y = -\frac{1}{\cos x}$ (c) $-\infty$

(d) $\begin{bmatrix} \cos x \\ 1/y \end{bmatrix}$ (e) $\frac{\partial}{\partial y} \sin x + \frac{\partial}{\partial x} \ln y$

Assignment 3.9: (a) Write the algorithm that uses gradient descent to find the vector $[x, y]^{\mathrm{T}}$ that minimizes the function $z = x \exp(-(x^2 + y^2))$. (b) Write a computer program that finds the minimizing x, y pair.

Assignment 3.10: Find the minimum of $y = 50 \sin(x) + x^2$ over $-10 \leq x \leq 10$. Use both gradient descent (GD) and simulated annealing (SA) to locate the minimum. This assignment intends to give students hands-on experience on how GD and SA work and how GD can get trapped at the local minimum but SA is capable of arriving at the global minimum regardless of the initial starting point.

1. Use MATLAB to plot this function. Visually observe the multiple local minima and the global minimum within the given interval of x.
2. Implement GD and test the code under the following parameter setups.
   (a) Pick a starting point at $x = 7$. What is the minimum?

(b) Pick a starting point at $x = 1$. What is the minimum?
(c) Change the step size (or learning rate) and see what kind of step size will help overpass the local minimum when $x = 7$.

3. Implement SA and test the code under the same parameter setups as above.
(a) Pick a starting point at $x = 7$. What is the minimum?
(b) Pick a starting point at $x = 1$. What is the minimum?
(c) Change the initial temperature and the change rate of temperature. Does it affect the result?

Bibliography

[3.1] R. Burden, J. Faires, and A. Reynolds. *Numerical Analysis*. Prindle, 1981.
[3.2] G. Dahlquist and A. Bjorck. *Numerical Methods*. Prentice-Hall, 1974.
[3.3] B. Gottfried and J. Weisman. *Introduction to Optimization Theory*. Prentice Hall, 1973.
[3.4] J. van Laarhoven and E. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel, 1988.

4 Images: Representation and Creation

Computers are useless. They can only give us answers.
– Pablo Picasso

4.1 Introduction

Since you have already had a course in image processing, it should not be necessary to describe how images are formed. Representations, however, are another matter. This chapter discusses various image representation schemes as well as a way of treating images as surfaces.

- (Section 4.2) In this section, we discuss mathematical representations both for the information contained in an image and for the ways in which images are stored and manipulated in a digital machine.
- (Section 4.3) In this section, we introduce a way of thinking about images – as surfaces with varying height – which we will find to be a powerful way to describe both the properties of images as well as operations on those images.

4.2 Image Representations

In this section, several ways to represent the information in an image are discussed. These representations include: iconic, functional, linear, probabilistic, and graphical representations. Note that in a digital image, the first dimension is *columns* and the second is *rows*. In a 3D digital image, the dimensions are *columns*, *rows*, and *frames*.

4.2.1 Iconic Representations (An Image)

An iconic representation of the information in an image is an image. Yeah, right; and a rose is a rose is a rose. When you see what we mean by functional, linear, and relational representations, you will realize we need a word[1] for a representation that is itself a picture. In the following, we briefly describe 2D, 3D, and range images.

- 2D images: The familiar 2D image is the brightness image, also called luminance image. These include photographs; the things you are used to calling "images" or "pictures."

[1] "Iconic" comes from the Greek word meaning picture.

These might be color or grayscale. (Be careful with the words "black and white," as that might be interpreted as "binary"). A shadow is a 2D binary image. We usually denote the brightness at a point $< x, y >$ as $f(x, y)$. Note: x and y could be either real numbers or integers. In the integer case, we are referring to discrete points in a sampled image. These points are called "pixels," short for "picture elements." In the case of real numbers we are usually thinking of the image as a function.

- 3D images: These data structures usually occur in medical images, like CT (computed tomography), MRI (magnetic resonance imaging), ultrasound, etc. In the typical 3D medical image, each pixel represents the density at a point. We usually denote the density at a point $< x, y, z >$ as $f(x, y, z)$. These three spatial dimensions are represented in a digital image by columns, rows, and frames.

  Movies are also 3D images, but the third dimension is not spatial, but temporal. It is represented by the frame number, corresponding to time.
- Range images: These are really 2D but the information they represent is interpreted in 3D space.

  In a range image, the value at each point represents a distance, usually the distance from the camera or the distance along a normal to a plane containing the camera. We may denote this distance at a point $< x, y >$ as $z(x, y)$. Suppose, for example, that the sensor is a laser, measuring time-of-flight. Then the "brightness" is actually proportional to the time required for a pulse of light to travel from the laser to a surface, bounce off, and return to a detector located at the source. In truth, such a sensor is measuring range as a function of two deflection angles, θ and ϕ and this function produces a *range image*.

  Figure 4.1 illustrates a brightness image and a range image of the same scene. In the range image, the apparent brightness can be seen increasing as distance from the scanner increases.

  To transform r, θ and ϕ coordinates to $z(x, y)$ is usually straightforward, and normally, any data you get (in this course anyway) will already have this correction performed to produce a *depth image*. The Microsoft Kinect™ can return a depth image.

  The bottom row of Figure 4.1 illustrates *random pseudocolor* renderings of the two images in the top row of Figure 4.1. To pseudocolor a grayscale image, one must first have an image with a fixed, finite number of brightness levels. The most common such scale is 0 to 255. Then, for each of the 256 possible brightnesses, a color is assigned. In the case of a random pseudocolor rendering, the colors are chosen at random.

  As can be seen here, the use of random pseudocolor allows the user to see aspects of an image that may not be apparent in the original. In this case, the noise shows up as pixel-sized dots of color. In Chapter 6 you will learn how to remove much of this noise.

4.2.2 Functional Representations (An Equation)

We can fit a function to any set of data points. When we measure a digital image, it always consists of a discrete and finite set of measurements, $f(x, y)$. By performing some operation such as *least-squares* data fitting, we can find a continuous function that best fits this set of data. We could thus represent an image, at least over a small area, by an equation such as a biquadratic:

$$z = ax^2 + by^2 + cxy + dx + ey + f \tag{4.1}$$

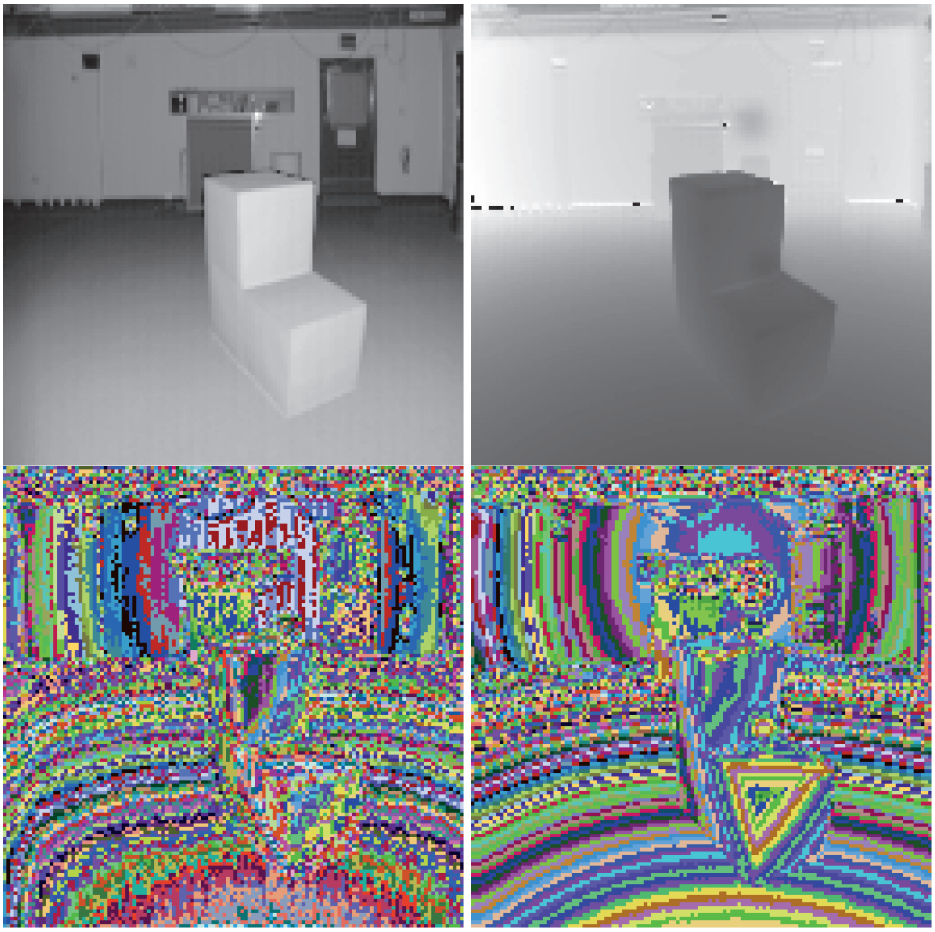


Figure 4.1 A brightness (top-left) and a range image (top-right) of the same scene. In this image, the brightness image is actually measured by the strength of the return laser beam. On the second line, pseudocolor renderings of the two images are presented. This way of viewing an image helps the student notice properties of the image, such as noise, which may not be apparent to the human's eye.

or a quadric:

$$ax^2 + by^2 + cz^2 + dxy + exz + fyz + gx + hy + iz + j = 0\ . \tag{4.2}$$

The form given in Eq. 4.1, in which one variable is defined in terms of the others, is often referred to as an *explicit* representation, whereas the form of Eq. 4.2 is an *implicit* representation.

The *zero set* of a function $f(x)$ is defined to be the set of points where $f(x) = 0$. The implicit representation may be equivalently represented in terms of the zero set, $\{(x, y, z) : f(x, y, z) = 0\}$. Implicit polynomials have some convenient properties. For example, consider a circle $x^2 + y^2 = R^2$, and a point (x_0, y_0) that is not in the zero set of $f(x, y)$. That is, since the set of points (x, y) that satisfy

$$f(x, y) = x^2 + y^2 - R^2 = 0 \tag{4.3}$$

is the zero set, if we substitute x_0 and y_0 into the equation for $f(x, y)$, we know we get a nonzero result (since we said this point is not in the zero set); if that value is negative, we know that the point (x_0, y_0) is inside the curve, otherwise, outside [4.1]. This inside/outside property holds for all closed curves (and surfaces) representable by polynomials.

4.2.3 Linear Representations (A Vector)

We can unwind the image into a vector, making vector-matrix operations very convenient in such a representation. For example, the 2×2 image $\left[\begin{smallmatrix} 5 & 10 \\ 6 & 4 \end{smallmatrix}\right]$ with brightness values as shown could be written as a vector $[\,5 \quad 10 \quad 6 \quad 4\,]^{\mathrm{T}}$. We will discuss this in more detail in section 5.3.

This representation of an image as a long vector in this way is called a *lexicographic* representation.

Another linear representation that will prove to be very important is that of the *shape matrix*. Suppose the boundary of some region has N points, and consider all the points on that boundary. Write them as 2-vectors of the form $[x, y]^{\mathrm{T}}$. Now collect all those points as columns of a single matrix

$$S = \begin{bmatrix} x_1 & x_2 & \dots & x_N \\ y_1 & y_2 & \dots & y_N \end{bmatrix}$$

That matrix is our simplest representation of a shape matrix. It will be encountered again in section 10.6.1.

4.2.4 Probabilistic Representations (A Random Field)

In Chapter 6, an image will be represented as the output of a random process that generates images. In that way we can make use of a powerful set of mathematical tools for estimating the best version of a particular image, given a measurement of a corrupted, noisy image. Such a representation will lead naturally to an optimization method that will reduce noise but preserve sharp edges.

4.2.5 Graphical Representations (A Graph)

A graph is a relational data structure. It consists of data elements, referred to as *vertices*, or *nodes* and relationships between vertices, referred to as *edges*.

Graphs may be completely described by sets. The set of vertices is a set of "interesting things," and the edges may be described by a set of ordered pairs. For example, let $G =< V, E >$ represent a graph, where the vertex set

$$V = \{a, b, c, d, e, f, g, h\}$$

and the edge set is

$$E = \{(a, b), (a, c), (a, d), (b, d), (c, d), (d, e), (f, g), (g, h)\}\ .$$

Graphs may also be represented pictorially, as illustrated in Figure 4.2 for this example.

In general, graphs may be directed. That is, the relations that are represented by edges may have a direction. Consider, for example, the relations "*ABOVE*" and "*ADJACENT*

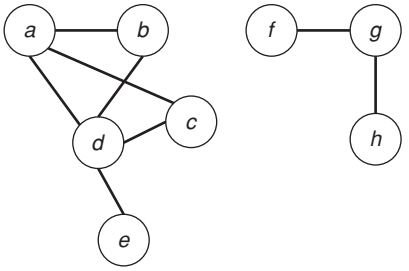


Figure 4.2 A graph containing two connected components. This graph contains two cliques of size 3, {a,b,d} and {a,c,d} and none of size 4.

TO". Clearly $ABOVE(A, B) \neq ABOVE(B, A)$, and so *ABOVE* would need to be represented by a directed graph, whereas the relation *ADJACENT TO* has no preferred direction. The pictorial version of a directed graph may be thought of as having arrowheads on the ends of the edges.

In the following, we define some important graph terminologies:

- The **degree** of a node is the number of edges coming into that node.
- A **path** between nodes v_0 and v_l is a sequence of nodes, $v_0, v_1, \ldots, v_l$ such that there exists an edge between v_i and v_{i+1} for $i = 0, 1, \ldots, l - 1$.
- A graph is **connected** if there exists a path between any two nodes.
- A **clique** (remember this word, you will see it again) is a subgraph in which there exists an edge between any two nodes.
- A **tree** is a graph that contains no loops. Tree representations have found application in image compression, image coding, and any application in which there is a hierarchical relation between data elements.

Graphs will show up several times in this book, as a means for describing relationships between regions and in segmentation. One application of graph representation is as follows: Every pixel is considered as a node in a graph. Edges may or may not exist between adjacent pixels. If an edge exists, then the two pixels lie in the same **connected component** of the graph. Consider the graph in Figure 4.2. This graph has 8 vertices, 8 edges, and two connected components. That is, there is a path along edges from a to d, or b to e, or f to h, etc. But there is no path from a to f.

The concept of connected components is easily related to pixels by defining a "label image," an iconic representation, isomorphic to the original image, in which each pixel contains the index/label information of the component to which it belongs. With this definition, we see that the image in Figure 4.3 contains two connected components (labeled 2 and 3) in addition to the background (labeled 1 and 4), as shown in Figure 4.4.

This *image graph* way of thinking may be used in segmentation in two ways: merging and cutting. A merging algorithm starts with the image graph initially containing no edges, and then adds edges to produce connected components. When thinking of pixels as nodes in the image graph, rather than referring to them as "pixel nodes," "pixel vertices," or other similar terminology, we simply continue to refer to them as "pixels."

Figure 4.3 An image containing two connected components.

A cutting algorithm begins with edges between all adjacent pixels, and then cutting edges to separate the image graph into distinct components. Examples of both methods will be described in Chapter 8.

4.2.6 The Adjacency Paradox and Hexagonal Pixels

As a sidenote to the subject of image representation, we discuss the issue of "adjacency" as well as the famous "adjacency paradox" in somewhat more depth. We also briefly describe an alternative sampling approach that requires hexagonally shaped pixels rather than the traditional square-shaped pixel. We discuss how this alternative sampling would completely resolve the adjacency paradox.

Two pixels are either adjacent to each other or not, right? In the sense of graphical representations for images, it would appear so. But pixels are (usually) rectangular, so they may be adjacent to pixels above, below, left, or right. In a graph representation, an edge of the graph runs between two nodes to indicate that they are adjacent. But how about diagonals? Let's look at that a bit more carefully.

We may define neighborhoods in a variety of ways, but the most common and most intuitive way is to say that two pixels are neighbors if they share a side (4-connected) or they share either a side or a vertex (8-connected). The neighborhood (or *aura*) of a pixel is the set of pixels that are neighbors (surprise!). The 4-neighbors of the center point are illustrated in Figure 4.5(b). The neighbors of a pixel are usually adjacent to that pixel, but there is no fundamental requirement that they be. We will see this again when we talk about "cliques."

```
11112221111111111111111111111
11122221111113333333333333111
11112221111113333333333333111
11112211111111111111333311111
11112211111111111111333331111
11112211111111111111333333111
11222222211111111113333333111
11224422211111111133333331111
11222222211111111111333111111
11112211111111111111111111111
```

Figure 4.4 The label image corresponding to Figure 4.3.

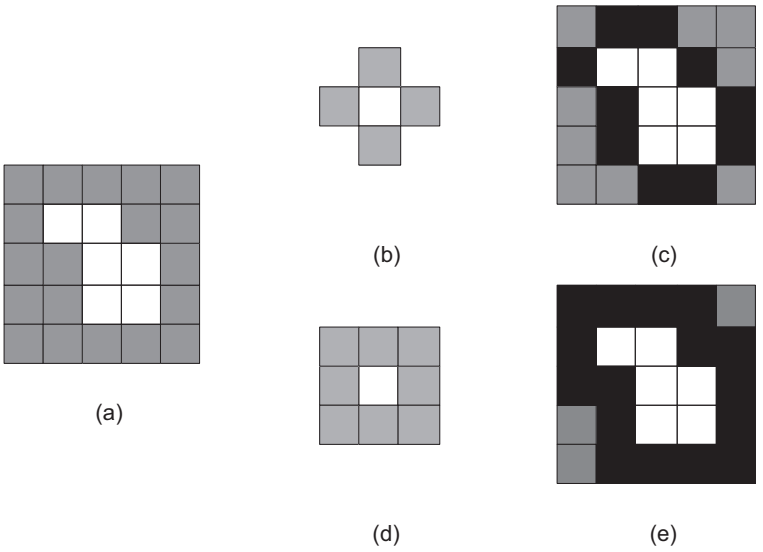


Figure 4.5 (a) Set A (shaded pixels) and set B (blank pixels). (b) The neighborhood relation for 4-connectedness – the shaded pixels are neighbors by definition of the center pixel. (c) The aura of the set of white pixels in (a) is given by the dark pixels using the 4-connected neighbor definition. (d) The neighborhood relation for 8-connectedness. (e) The aura of the set of white pixels in (a) using the 8-connected neighbor definition.

Digression: The Adjacency Paradox

In Figure 4.6, the foreground pixels are shown in black and the background in white. We have learned about 4-connected and 8-connectedness. Let's apply what we have learned. The foreground in this image is a ring, perhaps a very-low-resolution image of a washer. Is this ring closed? That is, can we start at one point and walk around the ring without retracing our steps, passing only from one pixel to a connected neighbor? (Incidentally, we just defined a property of connectivity called a *path* – see section 4.2.5). If we can walk all the way around this region like that, we say the region is closed. Use 4-connectedness. Is it closed? If you said no, good! In a 4-connected system, we cannot get from pixel *a* to pixel *b*. If it is not closed, it must be open. Right? If it is open, the inside and outside must be connected, right? But, using a 4-connected definition, the inside and outside are separate regions – a contradiction!

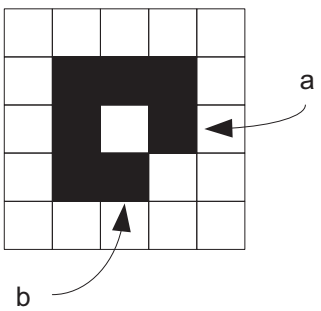


Figure 4.6 The adjacency paradox.

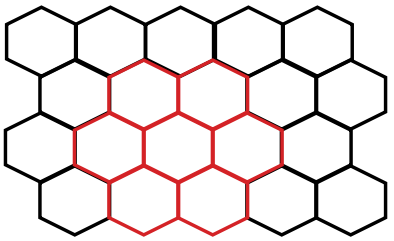


Figure 4.7 A neighborhood of hexagonally sampled pixels.

It seems we have to give up on 4-connectedness, since it leads to a paradox. Let's go to 8-connected. Now, the foreground is closed, right? But now, the inside and outside are also connected! How can that be? If the region is closed, the inside and outside must (logically) be separate. Therefore, 8-connectedness does not work either.

For this particular paradox, there is a fix, at least sort of a fix, which is to use one definition (8- or 4-connected) for the foreground and the other for the background. This fix works for binary images, but when we get to images with more than two levels of brightness, these discretization problems occur again. This example is presented to illustrate that in digital images, intuition is not always correct. There are lots of other examples of similar problems involving, for example, the measure of the perimeter of regions. Just be aware that, as in life, weird things happen and intuition does not always hold.

A Variation on Sampling: Hexagonal Pixels

Traditionally, electronic imaging sensors have been arranged in rectangular arrays mainly because an electron beam needed to be swept in a raster-scan way, and more recently because it is slightly more convenient to arrange semiconductor devices in a rectangular organization. However, as we saw above, rectangular arrays introduce an ambiguity in attempts to define neighborhoods. On the other hand, hexagonal pixels provide an ideal solution to the ambiguity problem.

We could easily imagine imaging sensors that use hexagonally organized pixel arrays. Hexagons are the minimum-energy solution when a collection of tangent circles with flexible boundaries are subject to pressure. The beehive is the best-known such naturally occurring organization, but many others occur too, including the organization of cones in the human retina. We see in Figure 4.7 that there are no connectivity paradoxes in hexagonal connectivity analysis: every pixel has exactly six neighbors, and the distance between each pixel and its immediate neighbors is the same along any of the six main directions of the hexagonal tessellation.

4.3 The Image as a Surface

In this section, we consider the problem of interpreting the image as a surface in a 3D space. Thinking of an image in this way will allow us to conceive of image properties as heights.

4.3.1 The Gradient

Consider the value of $f(x, y)$ as a surface in space, then the ordered triple $(x, y, f(x, y))$ describes this surface. For every point, (x, y), there is a corresponding value in the third dimension. It is important to observe that there is just one such f value for any (x, y) pair. If you draw a straight line in the x, y plane and ask yourself, "How does $f(x, y)$ change as I move along this line in x, y?", the question you have asked is the same as "What is the directional derivative at the point x, y along this particular direction?"

There is one special direction, the one that will maximize magnitude of the directional derivative, and we call it the *gradient*. The gradient vector has the special, very simple form: $G(x, y) = \left[\frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y} \right]^{\mathrm{T}}$.

One characteristic of gradients sometimes confuses students. The gradient is a 2-vector. It lies in the x, y plane. When you take a step in the direction of the gradient, you are moving in the direction in the plane (not uphill, out of the plane) such that f will rise as steeply as possible.

4.3.2 Isophotes

Consider the set of all points satisfying $f(x, y) = C$ for some constant C. If f represents brightness, then all the pixels in this set of points have the same brightness. We therefore refer to this set as an *isophote*, or a *level set*.

Theorem: At any image point, (x, y), the isophote passing through that point is perpendicular to the gradient. The proof is a homework.

4.3.3 Ridges

Think about a surface in space, written as $z(x, y)$. Now, z does not represent brightness, but elevation. We could easily think about it as a mountain. If we draw a geological contour map of this mountain, the lines in the map are lines of equal elevation. However, if we think of "elevation" denoting brightness, then the contour lines are isophotes. Stand at a point on this "mountain" and look in the direction the gradient is pointing, as in Figure 4.8; the direction you are looking is the way you would go to undertake the steepest ascent.

Note that the direction of the gradient is the steepest direction at that particular point. It does not necessarily point at the peak.

Let's climb this mountain by taking small steps in the direction of the local gradient.[2] What happens at the ridge line? How would you know you were on a ridge? How can you describe this process mathematically?

Think about taking steps in the direction of the gradient. Your steps are generally in the same direction until you reach the ridge; then, the direction radically shifts. So, one useful definition of a ridge is the locus of points that are **local maxima of the rate of change of gradient direction**. That is, we need to find the points where $|\frac{\partial f}{\partial v}|$ is maximized. Here, v

[2] A local maximum is any point that does not have a larger neighbor.

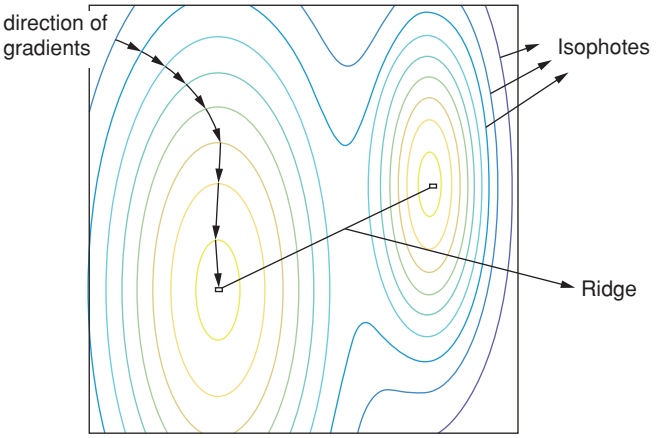


Figure 4.8 Illustration of direction of gradients, isophotes, and the ridge. Contour lines on an elevation map are equivalent to isophotes. The gradient vector at a point is perpendicular to the isophote at that point.

represents the direction of the gradient. In Cartesian coordinates,

$$\frac{\partial f}{\partial v} = \frac{2 f_x f_y f_{xy} - f_y^2 f_{xx} - f_x^2 f_{yy}}{(f_x^2 + f_y^2)^{3/2}} \,. \tag{4.4}$$

4.4 Assignments

Assignment 4.1: In Figure 4.7, we illustrate a totally different way to think about pixels. In that figure, instead of being rectangular, pixels are hexagonal, and each pixel has 6 neighbors instead of 4 or 8. If you had a camera with hexagonal pixels, would that help the adjacency paradox? Discuss.

Assignment 4.2: Suppose the image $f(x, y)$ is describable by $f(x, y) = x^4/4 - x^3 + y^2$. At the point $x = 1, y = 2$, which of the following is a unit vector that points along the isophote passing through that point?

(a) $\begin{bmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix}^{\mathrm{T}}$ (b) $\begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{bmatrix}^{\mathrm{T}}$ (c) $\begin{bmatrix} \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{bmatrix}^{\mathrm{T}}$ (d) $\begin{bmatrix} -2 & 4 \end{bmatrix}^{\mathrm{T}}$ (e) $\begin{bmatrix} 2 & 1 \end{bmatrix}^{\mathrm{T}}$ (f) $\begin{bmatrix} \frac{-2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix}^{\mathrm{T}}$

Bibliography

[4.1] R. Bajcsy and F. Solina. Three dimensional object recognition revisited. In *Proceedings of ICCV*, 1987.

Part II

Preprocessing

When the authors first started in Computer Vision, our intention was to write programs using the highest level, most elegant, most sophisticated Artificial Intelligence methods known. Of course, nothing worked.

Images are never what you expect. Real images all have noise. Even a perfect digital image is sampled, and sampling itself introduces a type of noise. Then there is blur and other distortions.

We begin this Part of the book by looking at the issue of finding edges in noisy images. That chapter will not only show how to find edges; it will introduce some other basic topics in image manipulation such as convolution.

Then, we move into removing noise. Blurring an image will certainly remove noise. Of course, it will remove image content also. The trick is to remove the noise but preserve image details, and that topic is discussed in the next chapter.

The final topic in this Part is morphology, a collection of nonlinear methods for erasing small details and closing gaps. The distance transform, a compact and useful representation for the information in an image, is also included in the morphology chapter.

5 Kernel Operators

You need boundaries . . . Even in our material creations, boundaries mark the most beautiful of places, between the ocean and the shore, between the mountains and the plains, where the canyon meets the river.

– Wm. Paul Young

5.1 Introduction

In this chapter, linear operations on images are investigated. The derivative, probably the most common linear operator, is considered first. That discussion is extended into edge detection, the result of applying derivative kernels on images. A variety of methods for accomplishing this objective are considered.

- (Section 5.2) We first define what makes a linear operator and explain the kernel operator, or sum of products, which is the type of linear operator that will appear most frequently in this book.
- (Section 5.3) We show in this section how to use vector representation of images to convert kernel operators to the matrix multiplication form in order to facilitate some analytical study.
- (Section 5.4) We focus on the discussion of derivatives, since it is probably the most common kernel operator used in Computer Vision applications. We discuss how to find the derivative kernel operators through the definition of derivatives, by function fitting, using vector representation of images, and taking derivatives of special blurring kernels.
- (Section 5.5) Applying derivative operators on images would result in edge images. We describe in this section the different types of edges and a popular edge detector, the Canny edge detector.
- (Section 5.6) The concept of scale space is explained and how the scale space is associated with the detection of edges at different scales.
- (Section 5.7) In Chapter 1, we briefly described the biological vision. Here, we revisit this issue and ask the question how human beings perform lower-level vision, i.e., edge detection. We introduce the Gabor filter, which is believed to well characterize the functionality of receptive fields in the visual cortex.
- (Section 5.8) We wrap up this chapter by performing an experimental study on the various types of derivative kernels in terms of their ability in detecting edges (both angle and magnitude) of different orientations.

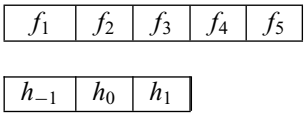


Figure 5.1 A one-dimensional image with five pixels and a one-dimensional kernel with three pixels. The subscript is the x-coordinate of the pixel.

5.2 Linear Operators

An operator $D(f)$ that operates on one image, f, to produce a second image, g, is said to be *linear* if

$$g = D(\alpha f_1 + \beta f_2) = \alpha D(f_1) + \beta D(f_2), \tag{5.1}$$

where f_1 and f_2 are images, α and β are scalar multipliers.

In the case that f is sampled, as digital images are, many authors choose to write f as a matrix, f_{ij}, instead of the functional notation $f(x, y)$. However, we prefer the x, y notation, for reasons that shall become apparent later. We will also find it more convenient to use a single subscript f_i, at several points later, but for now, let's stick with $f(x, y)$ and remember that x and y take on only a small range of integer values, e.g., $0 < x < 511$.

Think about a one-dimensional image named f with five pixels and another one-dimensional image, which we will call a *kernel* named h with three pixels, as illustrated in Figure 5.1. If we place the kernel down so that its center, pixel h_0, is over some particular pixel of f, say f_2, we get $g_2 = f_1 h_{-1} + f_2 h_0 + f_3 h_1$, which is a sum of products of elements of the kernel and elements of the image.

With that understanding, consider the common application to two-dimensional images. The general form of sum of products is calculated as follows,

$$g(x, y) = \sum_{\alpha} \sum_{\beta} f(x + \alpha, y + \beta) h(\alpha, \beta). \tag{5.2}$$

To better capture the essence of Eq. 5.2, let us write h as a 3×3 grid of numbers. Remember, $y = 0$ is at the TOP of the image and y goes up as you go down in the image[1]. Now imagine placing this grid down on top of the image so that the center of the grid is directly over pixel $f(x, y)$. Then each h value in the grid is multiplied by the corresponding point in the image. We will refer to the grid of h values henceforth as a "kernel." Assume for now that α and β take on only the values -1, 0, and 1, as shown in Figure 5.2. In this case, Eq. 5.2 expands to Eq. 5.3, a process we will come to call *application of a* 3×3 *kernel*.

$$\begin{aligned} g(x, y) = {} & f(x-1, y-1)h(-1, -1) + f(x, y-1)h(0, -1) \\ & + f(x+1, y-1)h(1, -1) + f(x-1, y)h(-1, 0) \\ & + f(x, y)h(0, 0) + f(x+1, y)h(1, 0) + f(x-1, y+1)h(-1, 1) \\ & + f(x, y+1)h(0, 1) + f(x+1, y+1)h(1, 1) \end{aligned} \tag{5.3}$$

[1] Note the order of the arguments here, x (column) and y (row). Sometimes the reverse convention is followed.

$h(-1,-1)$	$h(0,-1)$	$h(1,-1)$	$f(x-1,y-1)$	$f(x,y-1)$	$f(x+1,y-1)$
$h(-1,0)$	$h(0,0)$	$h(1,0)$	$f(x-1,y)$	$f(x,y)$	$f(x+1,y)$
$h(-1,1)$	$h(0,1)$	$h(1,1)$	$f(x-1,y+1)$	$f(x,y+1)$	$f(x+1,y+1)$

Figure 5.2 The kernel as a 3×3 grid (LEFT) to be applied to the image neighborhood centered at pixel $f(x, y)$ in the image (RIGHT). The kernel is located so that the center of the kernel falls over the x, y pixel of the image.

On the Direction of the Arguments: Convolution and Correlation

Let's restate two important equations: First the equation for a kernel operator, recopied from Eq. 5.2, and then the equation for two-dimensional, discrete convolution.[2]

$$g(x,y) = \sum_{\alpha}\sum_{\beta} f(x+\alpha, y+\beta)h(\alpha,\beta) \tag{5.4}$$

$$g(x,y) = \sum_{\alpha}\sum_{\beta} f(x-\alpha, y-\beta)h(\alpha,\beta) \tag{5.5}$$

The observant student will have noticed a discrepancy in order between the two equations above. In formal convolution, as given by the lower equation, the arguments reverse: the right-most pixel of the kernel (e.g., h_1 in Figure 5.1) is multiplied by the left-most pixel in the corresponding region of the image (e.g., f_1 in Figure 5.1). However, in the upper equation, we think of "placing" the kernel down over the image and multiplying corresponding pixels. If we multiply corresponding pixels, left-left and right-right, we have correlation, and this is usually what is meant by a *kernel operator*. There is, unfortunately a misnomer in much of the literature – both may sometimes be called "convolution." We advise the student to watch for this. In many publications, the term "convolution" is used when "sum of products" is meant. In order to avoid confusion, in this book, we will usually avoid the use of the word "convolution" unless we really do mean the application of the lower equation, and instead use the term "kernel operator," or "correlation," when that is what we mean. Note that if the kernel is symmetric (e.g., $\boxed{1}\boxed{2}\boxed{1}$), correlation and convolution give the same result.

5.3 Vector Representations of Images

In section 4.2.3, we briefly discussed the vector representation of an image. Here that discussion is extended in additional detail. Sometimes it is more convenient to represent the image in vector form such that the application of linear operators on images can be written in the form of matrix multiplication. Suppose we list every pixel in an image in raster scan order, as one long vector. For example, for the 4×4 image

$$f(x,y) = \begin{array}{|c|c|c|c|}\hline 1&2&4&1\\\hline 7&3&2&8\\\hline 9&2&1&4\\\hline 4&1&2&3\\\hline\end{array}$$

$$\mathbf{f} = [1\ 2\ 4\ 1\ 7\ 3\ 2\ 8\ 9\ 2\ 1\ 4\ 4\ 1\ 2\ 3]^{\mathrm{T}}.$$

[2] Mathematically, convolution and correlation differ in the left-right order of coordinates.

This is called the *lexicographic* representation. If we write the image in this way, each pixel may be identified by a single index, e.g., $\mathbf{f}_0 = 1, \mathbf{f}_4 = 7, \mathbf{f}_{15} = 3$, where the indexing starts with zero. Now suppose we want to apply the following kernel

−1	0	2
−2	0	4
3	9	1

to this image at point $x = 1, y = 1$, where (0, 0) is the upper-left pixel of the image. The lexicographic representation of this kernel is $\mathbf{k} = [-1\ 0\ 2\ -2\ 0\ 4\ 3\ 9\ 1]^{\mathrm{T}}$.

Point (1,1) in the image form corresponds to pixel $\mathbf{f}_5$ in the vector form. "Applying" a kernel means (usually) placing the center of the kernel over the pixel of interest in the image. In this example, we could accomplish this application of the kernel by taking the dot product of the vector $\mathbf{f}$ with the vector

$$\mathbf{h}_5 = [-1\ 0\ 2\ 0 \quad -2\ 0\ 4\ 0 \quad 3\ 9\ 1\ 0 \quad 0\ 0\ 0\ 0]^{\mathrm{T}}$$

As a second example, to determine what vector to use to apply this kernel at (2,1), we find

$$\mathbf{h}_6 = [0\ -1\ 0\ 2 \quad 0\ -2\ 0\ 4 \quad 0\ 3\ 9\ 1 \quad 0\ 0\ 0\ 0]^{\mathrm{T}}$$

Compare $\mathbf{h}_5$ at (1,1) and $\mathbf{h}_6$ at (2, 1). They are the same except for a rotation. We could perform the kernel operation on the entire image by constructing a matrix in which each column is one such $\mathbf{h}$. Doing so would result in a matrix such as the one illustrated below. By producing the product $\mathbf{g} = H^{\mathrm{T}}\mathbf{f}$, $\mathbf{g}$ will be the (lexicographic form of) result of applying kernel H on image $\mathbf{f}$, where H is the following matrix:

$$H = \begin{bmatrix}
\dots & -1 & 0 & \dots & 0 & \dots \\
\dots & 0 & -1 & \dots & 0 & \dots \\
\dots & 2 & 0 & \dots & 0 & \dots \\
\dots & 0 & 2 & \dots & 0 & \dots \\
\dots & -2 & 0 & \dots & 0 & \dots \\
\dots & 0 & -2 & \dots & -1 & \dots \\
\dots & 4 & 0 & \dots & 0 & \dots \\
\dots & 0 & 4 & \dots & 2 & \dots \\
\dots & 3 & 0 & \dots & 0 & \dots \\
\dots & 9 & 3 & \dots & -2 & \dots \\
\dots & 1 & 9 & \dots & 0 & \dots \\
\dots & 0 & 1 & \dots & 4 & \dots \\
\dots & 0 & 0 & \dots & 0 & \dots \\
\dots & 0 & 0 & \dots & 3 & \dots \\
\dots & 0 & 0 & \dots & 9 & \dots \\
\dots & 0 & 0 & \dots & 1 & \dots \\
 & \uparrow & \uparrow & & \uparrow & \\
 & \mathbf{h}_5 & \mathbf{h}_6 & & \mathbf{h}_{10} &
\end{bmatrix} \tag{5.6}$$

Note that the matrix H has a column for every pixel in the image. It is likely to be *very* large. If $\mathbf{f}$ is a typical image of 256×256 pixels, then H is $(256 \times 256) \times (256 \times 256)$ that is a large number (although still smaller than the US national debt). Do not worry about

the monstrous size of H. This form of image operators is useful for thinking about images and for proving theorems about image operators. It is a conceptual, not a computational, tool.

Finally, multiplication by a circulant matrix can be accomplished considerably faster by using the Fast Fourier transform; but we will say more about that at another time.

5.4 Derivative Estimation

Since the derivative is probably the most common linear operator, in this section we discuss four different ways to estimate derivatives. First, we describe an intuitive approach that uses the definition of the derivative to generate kernel operators. This is followed by a formal derivation of solving derivative kernels by function fitting. Based on vector representation of images, we introduce the concept of basis vectors and discuss some special basis vectors that describe edge-like characteristics. Finally, we elaborate on the method of constructing kernels by taking derivatives of special blurring kernels, making use of the linearity of kernel operators.

5.4.1 Using Kernels to Estimate Derivatives

Let us examine this concept via an example – approximating the spatial derivatives of the image, $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$. We recall from some dimly remembered calculus class,

$$\frac{\partial f}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{5.7}$$

which would suggest the kernel $\boxed{-1 \mid 1}$ could be used (for $\Delta x = 1$). But this kernel is not symmetric – the estimate at x depends on the value at x and at $x + 1$, but not at $x - 1$. That leaves the point of application difficult to define. Instead, a symmetric definition is better, such as

$$\frac{\partial f}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x}, \tag{5.8}$$

Δx cannot be smaller than 1, resulting in this kernel: $\boxed{-1/2 \mid 0 \mid 1/2}$. For notational simplicity, we write

$$1/2 \boxed{-1 \mid 0 \mid 1}. \tag{5.9}$$

A major problem with derivatives is noise sensitivity. To some extent, noise can be compensated for by taking the difference horizontally, and then averaging vertically – which produces the following kernel:

$$\frac{\partial f}{\partial x} = 1/6 \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \otimes f \tag{5.10}$$

where we have introduced a new symbol, $\otimes$ denoting the sum-of-products implementation described above. The literature abounds with kernels like this one. All of them combine the

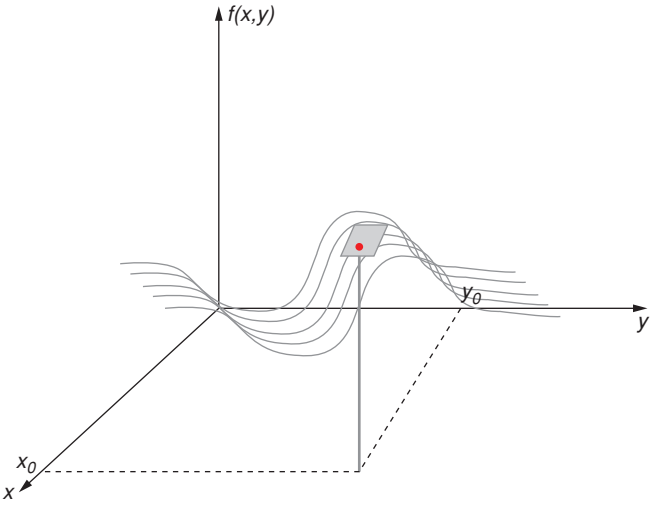


Figure 5.3 The brightness in an image can be thought of as a surface, a function of two variables. The surface is shown here as the smooth curves, and the flat surface represents the tangent plane at point x_0, y_0. The slopes of the tangent plane are the two spatial partial derivatives.

concept of estimating the derivative by differences, and then averaging the result in some way to compensate for noise. Probably the best known of these ad hoc kernels is the Sobel:

$$\frac{\partial f}{\partial x} = \frac{1}{8}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \otimes f. \tag{5.11}$$

The Sobel operator has the benefit of being center-weighted. That is, since the derivative is being estimated at the center point, it seems reasonable that the center row should count more than the rows above or below.

5.4.2 Derivative Estimation by Function Fitting

In this section, we present a fundamental approach to a data fitting problem, minimizing the sum-squared error. This general approach is presented in the context of a simple application: finding the plane that best fits a 2D set of points. However, it may find use in a wide variety of applications.

This approach presents an alternative way to make use of the continuous representation of an image $f(x, y)$. Think of the brightness as a function of the two spatial coordinates, and consider a plane that is tangent to that brightness surface at a point, as illustrated in Figure 5.3.

In this case, we may write the continuous image representation of a point on the tangent plane using the equation of a plane

$$f(x, y) = ax + by + c. \tag{5.12}$$

Then, the edge strength is described by the two numbers $\frac{\partial f}{\partial x} = a$, $\frac{\partial f}{\partial y} = b$. The rate of change of brightness at the point, (x, y), is represented by the gradient vector

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix}^{\mathrm{T}} = [a \;\; b]^{\mathrm{T}}. \tag{5.13}$$

The approach followed here is to find a, b, and c given some noisy, blurred measurement of f, and the assumption of Eq. 5.12. To find those parameters, first observe that Eq. 5.12 may be written as $f(x, y) = \mathbf{a}^{\mathrm{T}}\mathbf{x}$ where the vectors $\mathbf{a}$ and $\mathbf{x}$ are $\mathbf{a}^{\mathrm{T}} = [a \;\; b \;\; c]$ and $\mathbf{x}^{\mathrm{T}} = [x \;\; y \;\; 1]$.

Suppose we have measured brightness values $g(x, y)$ at a collection of points $\chi \subset Z \times Z$ (Z is the set of integers) in the image. Over that set of points, we seek the plane that best fits the data. To accomplish this objective, write the error as a sum-squared error function of the measurement, $g(x, y)$, and the (currently unknown) function $f(x, y)$.

$$E = \sum_{\chi} (f(x, y) - g(x, y))^2 = \sum_{\chi} (\mathbf{a}^{\mathrm{T}}\mathbf{x} - g(x, y))^2.$$

Expanding the square and eliminating the functional notation for simplicity, we find

$$E = \sum_{\chi} (\mathbf{a}^{\mathrm{T}}\mathbf{x})(\mathbf{a}^{\mathrm{T}}\mathbf{x}) - 2\mathbf{a}^{\mathrm{T}}\mathbf{x}g + g^2.$$

Remembering that for vectors $\mathbf{a}$ and $\mathbf{x}$, $\mathbf{a}^{\mathrm{T}}\mathbf{x} = \mathbf{x}^{\mathrm{T}}\mathbf{a}$, and taking the summation through, we have

$$E = \sum \mathbf{a}^{\mathrm{T}}\mathbf{x}\mathbf{x}^{\mathrm{T}}\mathbf{a} - 2\sum \mathbf{a}^{\mathrm{T}}\mathbf{x}g + \sum g^2 = \mathbf{a}^{\mathrm{T}} \left(\sum \mathbf{x}\mathbf{x}^{\mathrm{T}} \right) \mathbf{a} - 2\mathbf{a}^{\mathrm{T}} \sum \mathbf{x}g + \sum g^2.$$

Now we wish to find the $\mathbf{a}$ (the parameters of the plane) that minimizes E; so we may take derivatives and set the result to zero. (This is the derivative of a scalar with respect to a vector, so the result must be a vector.) We obtain

$$\frac{dE}{d\mathbf{a}} = 2\mathbf{a}^{\mathrm{T}} \left(\sum \mathbf{x}\mathbf{x}^{\mathrm{T}} \right) - 2\sum \mathbf{x}g = 0. \tag{5.14}$$

Let's call $\sum \mathbf{x}\mathbf{x}^{\mathrm{T}} = S$ (it is the "scatter matrix") and see what Eq. 5.14 means: consider a neighborhood χ that is symmetric about the origin. In that neighborhood, suppose x and y only take on values of -1, 0, and 1. Then

$$S = \sum \mathbf{x}\mathbf{x}^{\mathrm{T}} = \sum \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} [x \;\; y \;\; 1] = \begin{bmatrix} \sum x^2 & \sum xy & \sum x \\ \sum xy & \sum y^2 & \sum y \\ \sum x & \sum y & \sum 1 \end{bmatrix}$$

which, for the neighborhood described, is

$$\begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 9 \end{bmatrix}.$$

If you do not see where those values came from, note that the positive direction for y is down.

Here's a bit more detail on how you get these values. Look at the top left point, at coordinates $x = -1, y = -1$. At that point, $x^2 = (-1)^2 = 1$. Now look at the top middle point, at coordinates $x = 0, y = -1$. At that point, $x^2 = 0$. Do this for all 9 points in the neighborhood, and you obtain

$$\sum x^2 = 6.$$

Note this useful observation: If you make the neighborhood symmetric about the origin, all the terms in the scatter matrix that contain x or y to an odd power will be zero.

Also: a common mistake is to put a 1 in the lower right corner rather than a 9 – be careful!

Now we have the matrix equation

$$2\begin{bmatrix} a & b & c \end{bmatrix}\begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 9 \end{bmatrix} = 2\begin{bmatrix} \sum g(x,y)x \\ \sum g(x,y)y \\ \sum g(x,y) \end{bmatrix}.$$

We can easily solve for a:

$$a = \frac{1}{6}\sum g(x,y)x \approx \frac{\partial f}{\partial x}.$$

So, to compute the derivative from a fit to a neighborhood at each of the 9 points in the neighborhood, take the measured value at that point, $g(x, y)$, multiply by its x coordinate, and add them up. Let's write down the x coordinates in tabular form:

−1	0	1
−1	0	1
−1	0	1

.

That is precisely the kernel of Eq. 5.10, which we derived intuitively. Now, we have derived it formally. Isn't it satisfying when theory agrees with intuition? (Here, we did not multiply each term by 1/6, but we can simply factor the 1/6 out, and when we get the answer, we will just divide by 6).

This was accomplished by using an optimization method, in this case minimizing the squared error, to find the coefficients of a function $f(x)$ in an equation of the form $y = f(x)$, where f is polynomial. This form is referred to as an *explicit functional representation*.

One more terminology issue: In future discussions, we will use the expression *radius of a kernel*. The radius is the number of pixels from the center to the nearest edge. For example, a 3×3 kernel has a radius of 1. A 5×5 kernel has a radius of 2, etc. It is possible to design kernels that are circular, but most of the time, we use squares.

5.4.3 Basis Vectors for Images

In section 5.3, we saw that we could think of an image as a vector. If we can do that for an image, surely we can do the same thing for a small sub-image. Consider the 9-pixel neighborhood of a single point. We can easily construct the 9-vector that is the lexicographic representation of that neighborhood. In linear algebra, we learned that any vector could be represented as a weighted sum of basis vectors, and we will apply that same concept here.

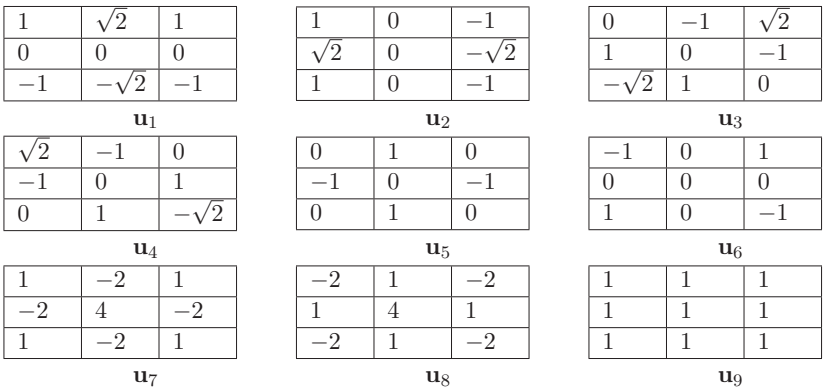


Figure 5.4 The Frei-Chen basis vectors.

Let's write the definition of a basis:

$$\mathbf{v} = \sum_{i=1}^{9} a_i \mathbf{u}_i$$

where $\mathbf{v}$ is now the vector representation of this 9-pixel neighborhood, the a_i are scalar weights, and the $\mathbf{u}_i$ are some set of *basis vectors*. But what basis vectors should we use? More to the point: What basis set would be useful? The set we normally use, the Cartesian basis, is

$$\begin{aligned} \mathbf{u}_1 &= [\,1\;0\;0\;0\;0\;0\;0\;0\;0\,]^{\mathrm{T}} \\ \mathbf{u}_2 &= [\,0\;1\;0\;0\;0\;0\;0\;0\;0\,]^{\mathrm{T}} \\ &\vdots \\ \mathbf{u}_9 &= [\,0\;0\;0\;0\;0\;0\;0\;0\;1\,]^{\mathrm{T}} \end{aligned} \tag{5.15}$$

which, while convenient and simple, does not help us at all here. Another basis would surely be more useful? Before we figure out what, remember that there is an infinity of possible basis vectors for this real-valued 9-space. With so many choices, we should be able to pick some good ones. To accomplish that, consider the role of the coefficients a_i. Recall that if some particular a_i is much larger than all the other a_i's, then $\mathbf{v}$ is "very similar" to $\mathbf{u}_i$.[3] Computing the a's then provides us a means to find which of a set of prototype neighborhoods a particular image most resembles.

Figure 5.4 illustrates a set of prototype neighborhoods developed by Frei and Chen [5.10]. Notice that neighborhood ($\mathbf{u}_1$) is negative below and positive above the horizontal center line, and therefore is indicative of a horizontal edge, a point where $\frac{\partial f}{\partial y}$ is large.

Now recall how to compute the projection a_i. The scalar-valued projection of a vector $\mathbf{v}$ onto a basis vector $\mathbf{u}_i$ is the inner product $a_i = \mathbf{v}^{\mathrm{T}}\mathbf{u}_i$.

Projection of two (suitably normalized) vectors is maximal if the two vectors are identical. This suggests that the projection operation, the inner product, can be used for

[3] A large projection means the two vectors are similar.

matching. But correlation (section 5.2) is, after all, just a sum of products, so projection doesn't resemble correlation – it *is* correlation.

5.4.4 A Kernel as a Sampled Differentiable Function

In Assignment 5.2, you are asked to construct a 5×5 derivative kernel by fitting a plane using the method derived in section 5.4.2. If you did it correctly, the kernel values increase as they are farther from the center. Does that make sense? Why should data points farther away from the point where we are estimating the derivative contribute more heavily to the estimate? Wrong! It is an artifact of the assumption we made that all the pixels fit the same plane. They obviously don't.

Here is a better way – weight the center pixel more heavily. You already saw this – the Sobel operator, Eq. 5.11 does it. But now, let's get a bit more rigorous. Let's blur the image by applying a kernel that is bigger in the middle and then differentiate. We have lots of choices for a kernel like that, e.g., a triangle or a Gaussian, but thorough research [5.25] has shown that a Gaussian works best for this sort of thing. We can write this process as

$$g_x \approx \frac{\partial (g \otimes h)}{\partial x},$$

where g is the measured image, h is a Gaussian, and g_x will be our new derivative estimate image. Now, a crucial point from linear systems theory: For linear operators D, and $\otimes$,

$$D(g \otimes h) = D(h) \otimes g. \tag{5.16}$$

Equation 5.16 means we do not have to do blurring in one step and differentiation in the next; instead, we can pre-compute the derivative of the blur kernel and simply apply the resultant kernel.

Let's see if we can remember how to take a derivative of a 2D Gaussian. (Did you forget it is a 2D function?)

A d-dimensional multivariate Gaussian has the general form

$$\frac{1}{(2\pi)^{d/2}|K|^{1/2}} \exp\left(-\frac{[\mathbf{x}-\mu]^{\mathrm{T}} K^{-1} [\mathbf{x}-\mu]}{2}\right) \tag{5.17}$$

where K is the covariance matrix, $|K|$ is its determinate, and μ is the mean vector. Since we want a Gaussian centered at the origin $\mu = \mathbf{0}$, and since we have no reason to prefer one direction over another, we choose K to be diagonal (isotropic):

$$K = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix} = \sigma^2 I. \tag{5.18}$$

For two dimensions, Eq. 5.17 simplifies to

$$h(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{[x\ y]^{\mathrm{T}}[x\ y]}{2\sigma^2}\right) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{5.19}$$

and

$$\frac{\partial h(x,y)}{\partial x} = \frac{-x}{2\pi\sigma^4} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right). \tag{5.20}$$

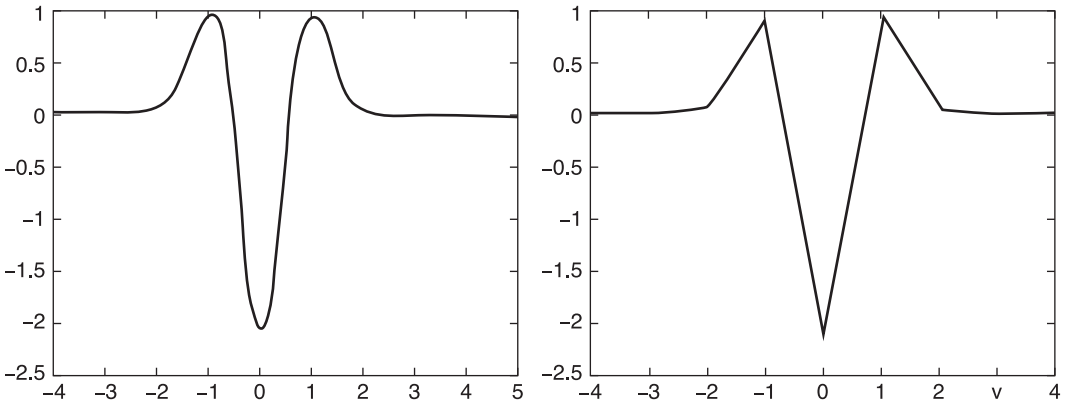


Figure 5.5 (a) The second derivative of a one-dimensional Gaussian centered at 0. (b) a three-point approximation.

If our objective is edge detection, we are done. However, if our objective is precise estimation of derivatives, particularly higher order derivatives, use of a Gaussian kernel, since it blurs the image, clearly introduces errors that can only be partially compensated for [5.30]. Nevertheless, this is one of the best ways to develop effective derivative kernels.

For future reference, a few of the derivatives of the one-dimensional Gaussian are given in Eq. 5.21. Even though there is no particular need for the normalizing 2π for most of our needs (it just ensures that the Gaussian integrates to one), we have included it. That way these formulae are in agreement with the literature. The subscript notation is used here to denote derivatives. That is, $G_{xx}(\sigma, x) = \frac{\partial^2}{\partial x^2} G(\sigma, x)$, where $G(\sigma, x)$ is a Gaussian function of x with mean of zero and standard deviation σ.

$$
\begin{aligned}
G(\sigma, x) &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) \\
G_x(\sigma, x) &= \frac{-x}{\sqrt{2\pi}\sigma^3} \exp\left(-\frac{x^2}{2\sigma^2}\right) \\
G_{xx}(\sigma, x) &= \left(\frac{x^2}{\sqrt{2\pi}\sigma^5} - \frac{1}{\sqrt{2\pi}\sigma^3}\right) \exp\left(-\frac{x^2}{2\sigma^2}\right) \\
G_{xxx}(\sigma, x) &= \frac{x}{\sqrt{2\pi}\sigma^5} \left(3 - \frac{x^2}{\sigma^2}\right) \exp\left(-\frac{x^2}{2\sigma^2}\right)
\end{aligned} \tag{5.21}
$$

Let's look in a bit more detail about how to make use of these formulae and their two-dimensional equivalents to derive kernels.

The simplest way to get the kernel values for the derivatives of a Gaussian is to simply substitute $x = 0, 1, 2$, etc., along with their negative values, which yields numbers for the kernel.

The first problem to arise is "what should σ be?" To address this question, we derive the elements of the kernel used for the second derivative of a one-dimensional Gaussian. The other derivatives can be developed using the same philosophy. Look at Figure 5.5 (a) and ask, "Is there a value of σ such that the maxima of the second derivative occur at

$x = -1$ and $x = 1$?" Clearly there is, and its value is $\sigma = 1/\sqrt{3}$. (You will prove this in Assignment 5.7.)

Given this value of σ, the values of the second derivative of a Gaussian can be computed at the integer points $x = \{-1, 0, 1\}$. At $x = 0$, we find $G_{xx}(\frac{1}{\sqrt{3}}, 0) = -2.07$, and at $x = 1$, $G_{xx}(\frac{1}{\sqrt{3}}, 1) = 0.9251$. So far, this isn't too difficult is it? Unfortunately, we are not done.

It is important that the elements of the kernel sum to zero. If they don't, then iterative algorithms like those described in Chapter 6 will not maintain the proper brightness levels over many iterations. The kernel also needs to be symmetric. That essentially defines the second derivative of a Gaussian. The most reasonable set of values close to those given, which satisfy symmetry and summation to zero are {1, -2, 1}. However, this does not teach us very much. Let's look at a 5×1 kernel and see if we can learn a bit more. We require:

- the elements of the kernel should approximate the values of the appropriate derivative of a Gaussian as closely as possible;
- the elements must sum to zero for a derivative kernel;
- the kernel should be symmetric about its center, unless you want to do special processing.

We can calculate the elements of a 5-element one-dimensional Gaussian, and if we do so, assuming $\sigma = \frac{1}{\sqrt{3}}$, for $x = \{-2, -1, 0, 1, 2\}$, we get $[0.0565, 0.9251, -2.0730, 0.9251, 0.0565]$. Unfortunately, those numbers do not sum quite to zero. It is important that the kernel values sum to zero, but not quite so important that the actual values be precise. So what do we do in a case like this? We use constrained optimization.[4] One strategy is to set up a problem to find a function that yields these values as closely as possible, but that integrates to zero. For more complex problems, the authors use Interopt [5.4], to solve numerical optimization problems, but you can solve this problem without using numerical methods. This is accomplished as follows. First, understand the problem (presented for the case of 5 points given above): We wish to find five numbers as close as possible to $[0.0565, 0.9251, -2.0730, 0.9251, 0.0565]$ that satisfy the constraint that the five sum to zero. Due to symmetry, we actually only have three numbers, which we will denote $[a, b, c]$. For notational convenience, introduce three constants $\alpha = 0.0565$, $\beta = 0.9251$, and $\gamma = -2.073$.

Thus, to find a, b, and c that resemble these numbers, we write the MSE form for the sum

$$H_o(a, b, c) = 2(a - \alpha)^2 + 2(b - \beta)^2 + (c - \gamma)^2. \tag{5.22}$$

and minimizing H_o will produce a, b, and c that are close to α, β, and γ. In order to further satisfy the "sum to zero" requirement of the kernel values, we need to add a constraint to H_o that requires $2a + 2b + c = 0$.

Using the concept of Lagrange multipliers, we can find the best choice of a, b, and c, by minimizing a different objective function[5]

$$H(a, b, c) = 2(a - \alpha)^2 + 2(b - \beta)^2 + (c - \gamma)^2 + \lambda(2a + 2b + c). \tag{5.23}$$

[4] This also gives us the opportunity to teach constrained optimization, an important and useful topic.
[5] H is the constrained version of H_o.

Table 5.1 Derivatives of a one-dimensional Gaussian when $\sigma = 1/\sqrt{3}$.

1st derivative, 3×1	[0.2420, 0.0, −0.2420] or [1, 0, −1]
1st derivative, 5×1	[0.1080, 0.2420, 0, −0.2420, −0.1080]
2nd derivative, 3×1	[1, −2, 1]
2nd derivative, 5×1	[0.07846, 0.94706, −2.05104, 0.94706, 0.07846]

A few words of explanation are in order for those students who are not familiar with constrained optimization using Lagrange multipliers. The term with the λ in front (λ is the Lagrange multiplier) is the constraint. It is formulated such that if it is exactly equal to zero, we should find the proper a, b, and c. By minimizing H, we will find the parameters that minimize H_o while simultaneously satisfying the constraint. To minimize H, take the partial derivatives and set them equal to zero:

$$\begin{aligned}
\frac{\partial H}{\partial a} &= 4a - 4\alpha + 2\lambda \\
\frac{\partial H}{\partial b} &= 4b - 4\beta + 2\lambda \\
\frac{\partial H}{\partial c} &= 2c - 2\gamma + \lambda \\
\frac{\partial H}{\partial \lambda} &= 2a + 2b + c
\end{aligned} \tag{5.24}$$

Setting the partial derivatives equal to zero and simplifying, we find the following set of linear equations:

$$\begin{aligned}
a &= \alpha - \frac{\lambda}{2} \\
b &= \beta - \frac{\lambda}{2} \\
c &= \gamma - \frac{\lambda}{2} \\
2a + 2b + c &= 0
\end{aligned} \tag{5.25}$$

which we solve to find the sets given in Table 5.1. In the case of the first derivative, 3×1, symmetry ensures the values always sum to zero, and therefore the integer values are just as good as the floating point ones.

We can proceed in the same way to compute the kernels to estimate the partial derivatives using Gaussians in two dimensions; one implementation of the first derivative with respect to x, assuming an isotropic Gaussian is presented in Figure 5.6. You will have the opportunity to derive others as homework problems.

0.0261	0	−0.0261
0.1080	0	−0.1080
0.0261	0	−0.0261

Figure 5.6 The 3×3 first derivative kernel, derived by using the derivative of a Gaussian.

5.4.5 Other Higher Order Derivatives

We have just seen how second or third derivatives may be computed using derivatives of Gaussians. Since the topic has come up, and you will need to know the terminology later, we define here two scalar operators that depend on the second derivatives: the *Laplacian* and the *quadratic variation*.

The Laplacian of brightness at a point (x, y) is

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{5.26}$$

whereas the quadratic variation (QV) of brightness is

$$\left(\frac{\partial^2 f}{\partial x^2}\right)^2 + \left(\frac{\partial^2 f}{\partial y^2}\right)^2 + 2\left(\frac{\partial^2 f}{\partial x \partial y}\right)^2. \tag{5.27}$$

The Laplacian may be approximated by several kernels, including the following three commonly used ones

$$\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}, \quad \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & -8 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}, \quad \begin{array}{|c|c|c|}\hline -1 & 2 & -1 \\ \hline 2 & -4 & 2 \\ \hline -1 & 2 & -1 \\ \hline \end{array} \tag{5.28}$$

and the quadratic variation can be estimated by three partial second derivative kernels,

$$\Delta_{xx} = \frac{1}{6}\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 1 & -2 & 1 \\ \hline 0 & 0 & 0 \\ \hline \end{array}, \Delta_{yy} = \frac{1}{6}\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 0 & -2 & 0 \\ \hline 0 & 1 & 0 \\ \hline \end{array}, \Delta_{xy} = 2\begin{array}{|c|c|c|}\hline -0.25 & 0 & 0.25 \\ \hline 0 & 0 & 0 \\ \hline 0.25 & 0 & -0.25 \\ \hline \end{array}. \tag{5.29}$$

Note that the Laplacian as illustrated in Eq. 5.28 is a 2D isotropic kernel, meaning that it does not have direction, as only one kernel is used.

The third kernel in Eq. 5.29 is frequently referred to as the *cross term* because it involves both x and y. The importance of the third (cross) term depends on the problem being solved. To see this, consider Figure 5.7, which illustrates the application of the Quadratic Variation without and with the cross term. The strength of the operator in the leftmost image, which was computed without the cross term, varies significantly as a function of edge orientation. The center image, however, uses all three terms in the QV, and has an almost perfectly uniform strength. Some students will realize that if $\frac{\partial^2 f}{\partial x^2}$ and $\frac{\partial^2 f}{\partial y^2}$ are in fact a basis for the second-order derivative space, then their sum should be invariant to gradient direction. Unfortunately, those basis terms are *squared* and then added, not just added, and the nonlinearity of the square introduces strength variations that require the cross term to compensate.

5.4.6 Introduction to Scale

So far, we have intentionally ignored the "scale" factor in the discussion of derivative kernels. In most Computer Vision applications the derivative of a Gaussian is used where the Gaussian parameter σ is a variable of the function, i.e., $G(x, y, \sigma)$, and σ is referred to as *scale*. Thus, the Laplacian is $G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma)$. Now that scale is a factor in the

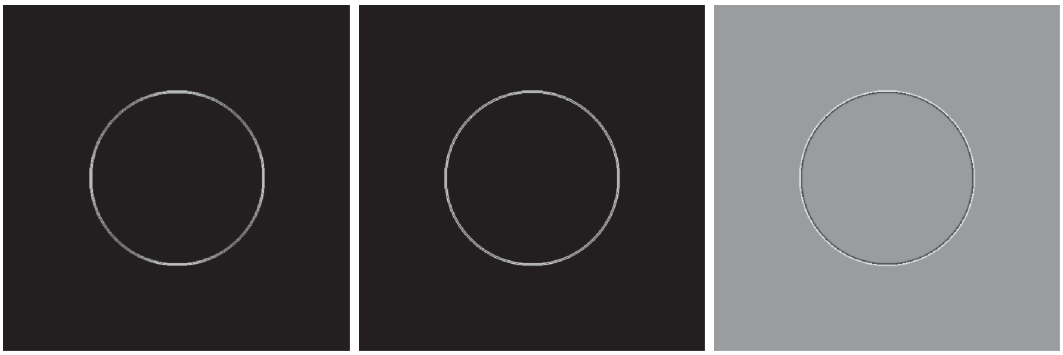


Figure 5.7 LEFT: Application of the first two terms of the QV to a circular step edge in brightness. CENTER: Application of the QV with all three terms to the same step edge. RIGHT: The Laplacian applied to the same step edge. The Laplacian has negative values, and the brightness is scaled to show black=most negative, gray=0, white = most positive.

derivative process, we must address the relationship between scale and zoom, the size of the kernel, and the scale of the function.

Suppose you have taken a 512×512 image, and there is some small thing in that image in which you are interested. You have great imaging software lying around, so you use one of those programs to draw a box around the item of interest and extract it. Such a process is illustrated in Figure 5.8.

Suppose the original image is 512×512 pixels and the object of interest is 128×128 pixels. One way to compute those pixels is: for every 4×4 square block of pixels in the original image, produce a single pixel in the smaller image. Simply averaging of the 16 original pixels produces an image in which the information is blurred, but the new image is small, as illustrated in the center of Figure 5.8. But, suppose your customer insists that the lower resolution image, which is now 128×128 still be 512×512. After doing that, the new image is noticeably blurred.

Figure 5.8 A 512×512 image was windowed, cropped, and a small region selected. Then the resultant image was zoomed again, producing a blurred image.

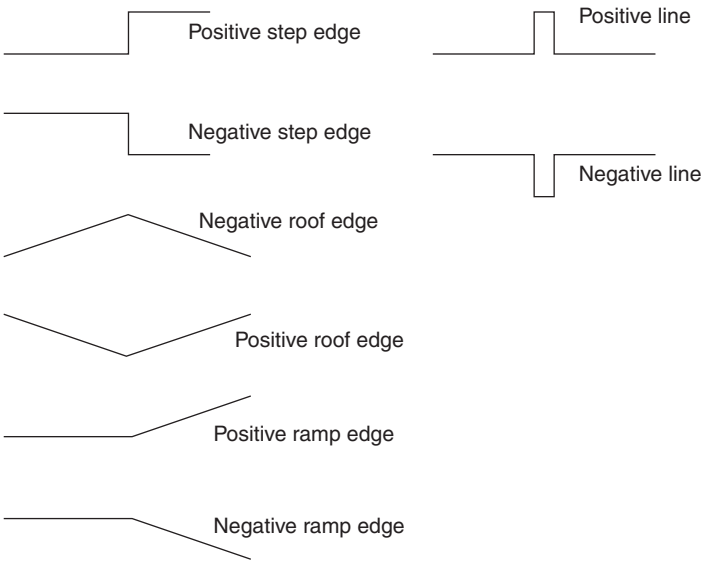


Figure 5.9 Brightness as a function of distance for several commonly occurring edges. Note that the term "positive" or "negative" generally refers to the sign of the first derivative at the edge point. A *line* can be defined as a positive edge followed immediately by a negative edge.

You can probably see the blur in the zoomed image, resulting from the smaller image (lower scale being interpolated to a larger image). Thus, we see a relationship between blur and scale. In scale space, as we will discuss in section 5.6, zooming in produces a smaller viewing area, with more detail – smaller scale, whereas zooming out produces a larger viewing area, but individual image features consist of fewer pixels, effectively blurred – larger scale. So one meaning of "scale" is the degree of blur, and a second is the relative size of the feature. Several additional details of this simple idea will come up later in this book.

5.5 Edge Detection

Edges are areas in the image where the brightness changes suddenly; where the derivative (or more correctly, some derivative) has a large magnitude. A common way to find edges is to start by applying the derivative kernels on the image. We can categorize edges as step, roof, or ramp [5.19], as illustrated in Figure 5.9.

We have already seen (twice) how application of a kernel such as that of Eq. 5.10 approximates the partial derivative with respect to x. Similarly, the kernels of Eq. 5.30 estimate $\frac{\partial f}{\partial y}$. Depending on which direction you have chosen as positive y, you may select the left kernel (positive y is pointing down) or the right kernel (positive y is pointing up).

$$h_y = \frac{1}{6}\begin{array}{|c|c|c|}\hline -1 & -1 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \text{ or } \frac{1}{6}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array}. \tag{5.30}$$

In the literature, some other forms have appeared, and you should know about some of them for historical reasons. Below are the two derivatives estimates we have already seen, referred to as the x- and y-versions of the "Sobel operator."

$$h_y = \frac{1}{8}\begin{array}{|c|c|c|}\hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline\end{array} \qquad h_x = \frac{1}{8}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline\end{array}. \tag{5.31}$$

IMPORTANT (this will give you trouble for the entire semester, so you may as well start now): In software implementations, the positive y direction is *down*! This results from the fact that scanning is top-to-bottom, left-to-right. So pixel (0,0) is the upper left corner of the image. Furthermore, numbering starts at zero, not one. We find the best way to avoid confusion is to never use the words "x" and "y" in writing programs, but instead use "row" and "column" remembering that now 0 is on top.

In this book, we will sometimes use conventional Cartesian coordinates in order to get the math right, and to further confuse the student (which is, after all, what professors are there for).

Having cleared up that muddle, let us proceed. Given the gradient vector

$$\nabla f = \left[\frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y}\right]^T \equiv \left[f_x \;\; f_y\right]^{\mathrm{T}}, \tag{5.32}$$

we are interested in its magnitude

$$|\nabla f| = \sqrt{f_x^2 + f_y^2} \tag{5.33}$$

(which we will call the "edge strength") and its direction

$$\angle \nabla f = \tan^{-1}\left(\frac{f_y}{f_x}\right). \tag{5.34}$$

One way to find an edge in an image is to compute the "gradient magnitude" image – that is, the image in which the brightness at each pixel is the magnitude of the gradient vector calculated at that point. Then, that value is thresholded to create an image that is zero for background and some large number (either 1 or 255) for edges. Go try it: do Assignment 5.4.

What did you learn from those experiments? Clearly, several problems arise when we try to find edges by simple kernel operations. We discovered that no matter what, noisy images result in edges that are:

- too thick in places
- missing in places
- extraneous in places

That is just life – we cannot do any better with simple kernels, but we will address more sophisticated methods in the next chapter.

5.5.1 The Canny Edge Detector

As you may have guessed, there are approaches to finding edges in images other than simply thresholding a derivative. We want to know the location of the edge more precisely than simple gradient thresholds reveal. Tagare and deFigueiredo [5.28] (see also [5.2]) generalize the process of edge detection as

1. The input is convolved with a filter that smoothly differentiates the input and produces high values at and near the location of the edge. The output $g(x)$ is the sum of the differentiated step edge and noise.
2. A decision mechanism isolates regions where the output of the filter is significantly higher than that due to noise.
3. A mechanism identifies the zero crossing in the derivative of $g(x)$ in the isolated region and declares it to be the location of the edge.

Two methods that have acquired a substantial reputation in this area are the so called "Canny edge detector" [5.6] and the "facet model" [5.11]. Here, we only describe the Canny edge detector.

This edge detection algorithm begins with finding estimates of the gradient magnitude at each point. Canny uses 2×2 rather than 3×3 kernels as we have used, but it does not affect the philosophy of the approach. Once we have estimates of the two partial derivatives, we use Eqs. 5.33 and 5.34 to calculate the magnitude and direction of the gradient, producing two images, $M(x, y)$ and $\Theta(x, y)$. We now have a result that easily identifies the pixels where the magnitude of the gradient is large. That is not sufficient, however, as we now need to thin the magnitude array, leaving only points that are maxima, creating a new image $N(x, y)$. This process is called *nonmaximum*[6] *suppression* (NMS).

NMS may be accomplished in a number of ways. The essential idea, however, is as follows. First, initialize $N(x, y)$ to $M(x, y)$. Then, at each point, (x, y), look one pixel in the direction of the gradient and one pixel in the reverse direction. If $M(x, y)$ (the point in question) is not the maximum of these three, set its value in $N(x, y)$ to zero. Otherwise, the value of N is unchanged.

After NMS, we have edges that are properly located and are only one pixel wide. These new edges, however, still suffer from the problems we identified earlier – extra edge points due to noise (false hits) and missing edge points due either to blur or to noise (false misses). Some improvement can be gained by using a *dual-threshold* approach. Two thresholds are used, τ_1 and τ_2, where τ_2 is significantly larger than τ_1. Application of these two different thresholds to $N(x, y)$ produces two binary edge images, denoted T_1 and T_2, respectively. Since T_1 was created using a lower threshold, it will contain more false hits than T_2. Points in T_2 are therefore considered to be parts of true edges. Connected points in T_2 are copied to the output edge image. When the end of an edge is found, points are sought in T_1 that could be continuations of the edge. The continuation is extended until it connects with another T_2 edge point, or no connected T_1 points are found.

[6] This is sometimes written using the plural, nonmaxima suppression. The expression is ambiguous. It could be a suppression of every point that is not a maximum, or suppression of all points that are not maxima. We choose to use the singular.

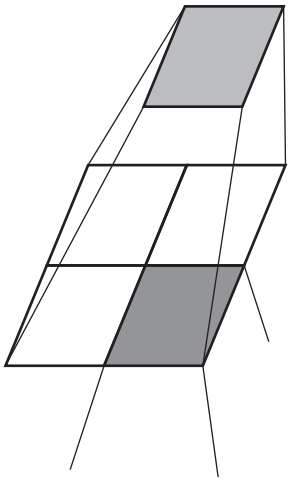


Figure 5.10 A pyramid is a data structure that is a series of images, in which each pixel is the average of four pixels at the next lower level.

In [5.6], Canny also illustrates some clever approximations that provide significant speed-ups.

5.6 Scale Space

In section 5.4.6, we briefly introduced the concept of scale. Here, we elaborate on the process of constructing a scale space as well as some important properties of scale space.

"Scale Space" is an extension to the concept of image pyramids, first used in picture processing by Kelly [5.18] and later extended in a number of ways (see [5.5, 5.7, 5.26, 5.27] and many others).

5.6.1 Pyramids

In a pyramid, a series of representations of the same image are generated, each created by a 2:1 subsampling (or averaging) of the image at the next higher level (Figure 5.10).

In Figure 5.11, a Gaussian pyramid is illustrated. It is generated by blurring each level with a Gaussian prior to 2:1 subsampling. An interesting question should arise as you look at this figure. Could you, from all the data in this pyramid, reconstruct the original image? The answer is "No, because at each level, you are throwing away high frequency information."

Although the Gaussian pyramid alone does not contain sufficient information to reconstruct the original image, we could construct a second pyramid to provide the additional information. To do that, we use a *Laplacian pyramid*, constructed by computing a similar representation of the image using the Laplacian operator. That process preserves the high frequency information. This is illustrated in Figure 5.12. Combining the two pyramid representations allows reconstruction of the original image.

Figure 5.11 A Gaussian pyramid, constructed by blurring each level with a Gaussian and then 2:1 subsampling. This way of viewing a pyramid emphasizes the fact that we should think of a pyramid as a three-dimensional data structure.

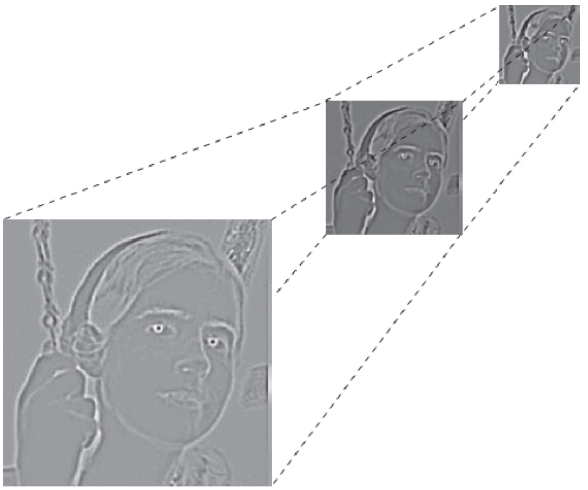


Figure 5.12 This Laplacian pyramid is actually computed by a difference of Gaussians. Also see section 11.3.2. This way of viewing a pyramid emphasizes the fact that each higher-level image has fewer pixels.

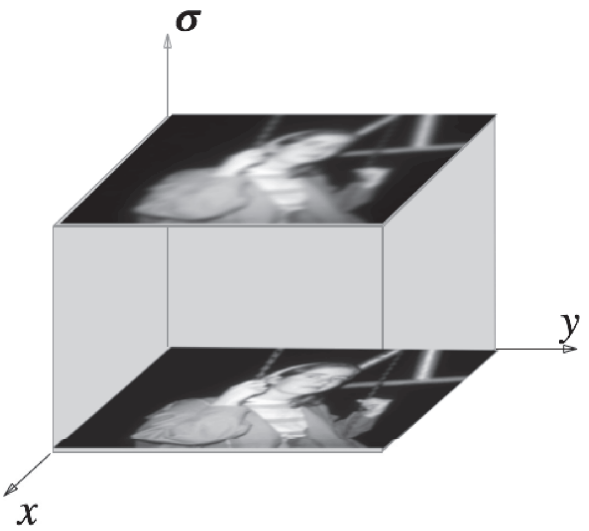


Figure 5.13 In a *stack scale space*, the original image is at the bottom, and each higher level is increasingly blurred, using a Gaussian blur with scale of σ.

5.6.2 Scale Space without Resampling

In a modern (stacked) scale space representation the concept that each level is a blurring of the previous level is preserved, but no subsampling is done – each level is the same size as the previous level, but more blurred. Clearly, at high levels of scale, i.e., σ is large, only the largest features are visible. This is illustrated in Figure 5.13.

Here is a *gedankenexperiment* for you: Take an image, blur it with a Gaussian kernel of standard deviation 1.

$$f(x, y, 1) = G(x, y, 1) \otimes f(x, y) \tag{5.35}$$

where

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}. \tag{5.36}$$

This operation produces a new image. Call that image 1. Now, blur the original image with a Gaussian kernel of standard deviation 2. Call that image 2. Continue until you have a set of images that you can think of as stacked, and the "top" image is almost blurred away. We say the top image is a representation of the image at "large scale."[7] This stack of images is referred to as a *scale-space representation*. Clearly, since we are not required to use only integer values of the standard deviation, we can create scale space representations with as much resolution in scale as desired. The essential premise of scale space representations is that certain features can be tracked over scale, and how those features vary with scale tells something about the image.

A scale space has been formally defined [5.22, 5.23] as having the following properties:

- All signals should be defined on the same domain (no pyramids).
- Increasing values of the scale parameter should produce coarser representations.

[7] A good way to remember large and small scale is that at large scale, only large objects may be distinguished.

- A signal at a coarser level should contain less structure than a signal at a finer level. If one considers the number of local extrema as a measure of smoothness, then the number of extrema should not increase as we go to coarser scale. This property is called *scale-space causality*.
- All representations should be generated by applications of a kernel operator to the original image.

The last property is certainly debatable, since it formally requires a linear, space-invariant operator. One interesting approach to scale space that violates this requirement is to produce a scale space by using grayscale morphological smoothing with larger and larger structuring elements [5.15]. You could use scale space concepts to represent texture or even a probability density function, in which case your scale-space representation becomes a clustering algorithm [5.21]. We will see applications of scale representations as we proceed through the course.

One of the most interesting aspects of scale space representations is the behavior of our old friend, the Gaussian. The second derivative of the Gaussian (in two dimensions, the Laplacian of Gaussian: LOG) has been shown [5.24] to have some very nice properties when used as a kernel. The zero crossings of the LOG are good indicators of the location of an edge.

Scale Space Causality

One might ask, "Is the Gaussian the best smoothing operator to use to develop a kernel like this?" Said another way: we want a kernel whose second derivative never generates a new zero crossing as we move to larger scale. In fact, we could state this desire in the following more general form:

Let our concept of a "feature" be a point where some operator has an extreme, either maximum or minimum. The concept of scale-space causality says that as scale increases, as images become more blurred, new features are never created. The Gaussian is the *only* kernel (linear operator) with this property [5.2, 5.3].

This idea of scale-space causality is shown in the following example. Figure 5.14 illustrates the brightness profile along a single line from an image, and the scale space created by blurring that single line with one-dimensional Gaussians of increasing variance. In Figure 5.15, we see the Laplacian of the Gaussian, and the points where the Laplacian changes sign. The features in this example, the zero-crossings (which are good candidates for edges) are indicated in the right image. Observe that as scale increases, new feature points (in this case, zero-crossings) are never created. As we go from top (low scale) to bottom (high scale), some features disappear, but no new ones are created.

One obvious application of this idea is to identify the important edges in the image first. We can do that by going up in scale, finding those few edges, and then tracking them down to lower scale.

5.7 So How Do People Do It?

Two neurophysiologists, David Hubel and Thorsten Wiesel [5.13, 5.14], stuck some electrodes in the brains – specifically the visual cortex – first of cats and later of monkeys.

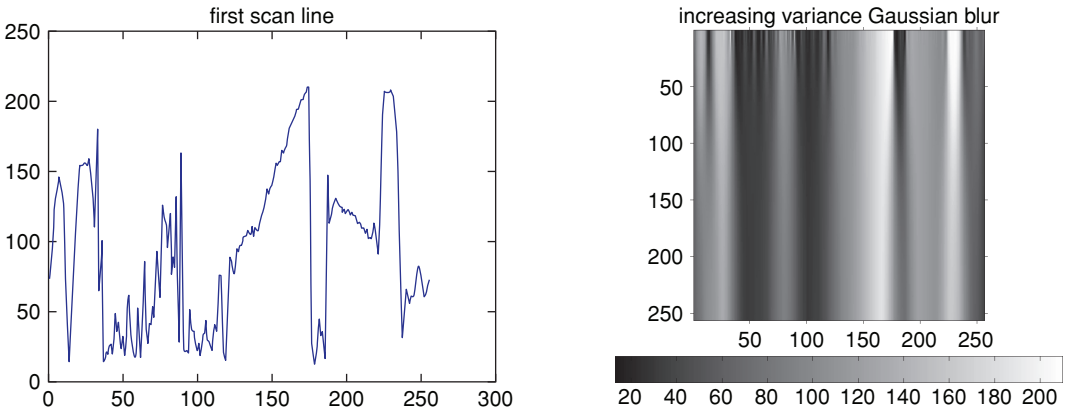


Figure 5.14 Left: brightness profile of a scanline through an image. Right: scale-space representation of that scanline. Scale increases toward the bottom, so no new features should be created as one goes from top to bottom.

While recording the firing of neurons, they provided the animal with visual stimuli of various types. They observed some fascinating results: first, there are cells that fire only when specific types of patterns are observed. For example, a particular cell might only fire if it observed an edge, bright to dark, at a particular angle. There was evidence that each of the cells they measured received input from a neighborhood of cells called a *receptive field*. There were a variety of types of receptive fields, possibly all connected to the same light detectors, which were organized in such a way as to accomplish edge detection and other processing. Jones and Palmer [5.16] mapped receptive field functions carefully and confirmed [5.8, 5.9] that the function of receptive fields could be accurately represented by Gabor functions, which have the form of Eq. 5.39.

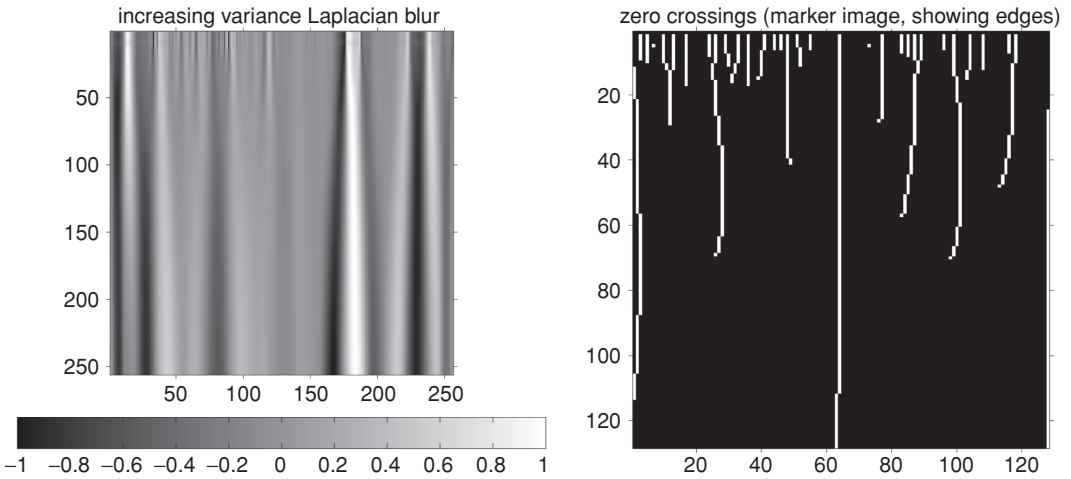


Figure 5.15 Laplacian of the scale space representation (Left), and the zero crossings of the Laplacian (Right). Since this is one-dimensional data, there is no difference between the Laplacian and the second derivative.

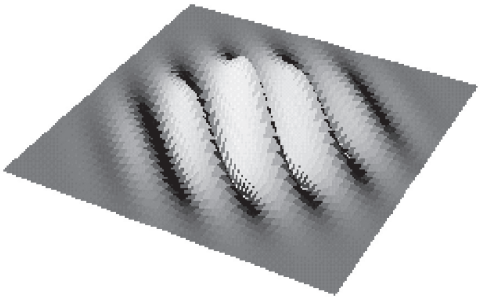


Figure 5.16 A two-dimensional Gabor filter. Note that the positive/negative response looks a lot like the fourth derivative of a Gaussian.

The Gabor is the product of a Gaussian and a sinusoid, where the Gaussian provides center weighting, and the sinusoid provides sensitivity to edges.

Despite all the parameters, the Gabor is relatively easy to use. It has gained popularity for three reasons. First, evidence from biology suggests the Gabor is a relatively good representative for the response of neurons in the visual cortex to edges in an image. Second, the Gabor is relatively easy to use as a directional derivative operator with parameterizable orientation. Furthermore, the selection of parameters allows the user to specify not only the scale but also the type of edge/line that the filter selects.

The Gabor filter, applied to a point, x, y, in an image, computes

$$x' = x\cos\theta + y\sin\theta \tag{5.37}$$

$$y' = -x\sin\theta + y\cos\theta \tag{5.38}$$

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right)\cos\left(2\pi\frac{x'}{\lambda} + \psi\right). \tag{5.39}$$

where the normal to the parallel stripes has the angle θ, λ is the variable that determines the wavelength of the sinusoid, γ is the aspect ratio (the ratio of length to width of the operator), ψ is the phase, σ is the standard deviation of the Gaussian (the scale).

If you are using the IFS library, the Gabor is implemented by using the IFS function `flGabor`.

Figure 5.16 illustrates a Gabor filter.

The Gabor filter, described in this way (you can choose to use complex exponentials instead of a cosine), has considerable flexibility. For example, if $\psi = 0$, the cosine is centered at the origin, and we produce a filter that looks (in 1D) very similar to the top curve in Figure 5.17. This filter looks almost exactly like $-\frac{\partial^4 f}{\partial x^4}$ and could be considered a fourth derivative operator, or a line detector, since a line is really simply two edges back-to-back. If, however, $\psi = \frac{\pi}{2}$, the cosine turns into a sine, and we have a rising step edge detector. Similarly, $\psi = -\frac{\pi}{2}$ will produce a falling edge detector.

The exponential factor of Eq. 5.39 is a two-dimensional Gaussian whose isophotes form ellipses with major and minor axes aligned with the x and y axes.

The following interesting observations have been made [5.20] regarding the values of the parameters in Eq. 5.39, when those parameters are actually measured in living organisms:

- The aspect ratio, γ, of the ellipse is 2:1.

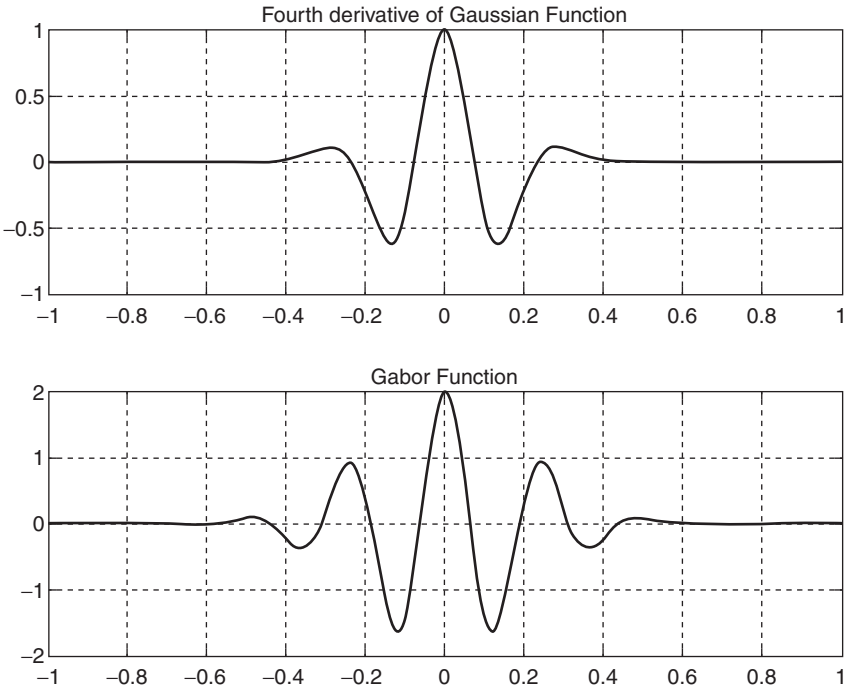


Figure 5.17 Comparison of a section through a Gabor filter and a similar section through a fourth derivative of a Gaussian.

- The plane wave created by the sinusoid tends to propagate along the short axis of the ellipse.
- The half-amplitude bandwidth of the frequency response is about 1 to 1.5 octaves along the optimal orientation.

So do we have Gabor filters in our brains? Or Gabor-like wavelet generators? We don't know about *your* brain, but Young [5.31] has looked at the stimulus-response characteristics of mammalian retina and observed that those same receptive fields that are so nicely modeled with Gabor filters or wavelets may equally well be described in terms of kernels called "difference of offset Gaussians," which is essentially the LOG with an additive offset. Figure 5.17 illustrates a section through a Gabor and a fourth derivative of a Gaussian. You can see noticeable differences, the principal one being that the Gabor goes on forever, whereas the fourth derivative has only three extrema. However, to the precision available to neurophysiology experiments, they appear to be the same. The problem is simply that the measurement of the data is not sufficiently accurate, and it is possible to fit a variety of curves to it.

5.8 Performance of Digital Gradient Detectors

Image gradients play a strong role in analysis of images. For example, the direction of the gradient is the direction that produces the largest infinitesimal increase in brightness for an

infinitesimal step in that direction. Unfortunately, in processing of digital images, the word "infinitesimal" must be translated as "equal to one." The approach to finding the gradient then becomes simply a threshold on the magnitude found by Eq. 5.33.

The problem is, of course, the operators that estimate the vertical and horizontal components of the gradient do not do so without error. When we encountered this in an investigation of a totally unrelated phenomenon, we said, "This is old stuff. Surely there is a paper that quantified the performance of gradient operators written back in the seventies." After a search of the complete paper copies of the *IEEE Transactions on Pattern Recognition and Machine Intelligence*, since the inception of publication, the most recent paper we found on the topic of how well gradient operators perform was written in 2014 [5.1]. Maybe there is still something to be learned about this topic.

In that paper [5.1], Alfaraj and Wang addressed how the kernel used affects the quality of the estimate of the gradient vector. This section elaborates on this topic and is primarily experimental. The performance of the simple 1×3 derivative, the simple 5×5, the Sobel, the Gabor, and the derivative of a Gaussian are compared.

Two questions are addressed:

- how consistent is this operator in its response to edges at particular angles, and
- how accurate is use of the 0-degree and 90-degree versions of the operator together (Eq. 5.34) in estimating the orientation of a particular edge segment.

5.8.1 The Directional Derivative

In the first set of tests, a perfect synthetic vertical edge passing through the center of the image is used.

A second set of tests is identical to the first except for the fact that a small ($\sigma \approx 1$) blur is applied to the input image, resulting in a blur of the ideal step edge. The results from blurred images are presented beside the results from ideal images in the studies below.

There is no center pixel for an image with an even number of rows or columns. For that reason, all the images used in these experiments are 129×129. Without loss of generality the edge is defined to be between a uniform region of brightness 0.0 and a region of brightness 1.0.

The directional derivative is defined to be the rate of change of brightness in a particular direction. Most gradient operators commonly used in Computer Vision only come in two directions, horizontal and vertical. The Gabor filter is a commonly used exception to this, and will be discussed more below.

In all figures below, the edge orientation is defined to be the angle θ illustrated in Figure 5.18.

Simple 1×3 Filters

The smallest symmetric difference operators, are illustrated below, with the horizontal on the left.

$$\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline \end{array} \qquad \begin{array}{|c|}\hline -1 \\ \hline 0 \\ \hline 1 \\ \hline \end{array} . \tag{5.40}$$

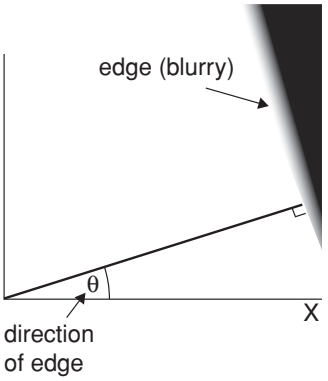


Figure 5.18 The orientation of an edge is defined to be the angle the normal to the edge makes with the x axis.

Applying this operator to a variety of edges produces the graph shown in Figure 5.19. On the left, the figure illustrates the result when applied to ideal step edges; on the right, to edges that are slightly blurred. Observe that when applied to a perfect step edge, these two operators produce a binary output. The vertical operator has only two possible outputs, and the horizontal likewise has only two. So the use of a pair of 1×3 differences can only distinguish four orientations – very low resolution in orientation space. Application to a blurred step edge is better, since the blurring operation spreads the information about the edge over a larger area.

Sobel Kernels

Both of the well-known Sobel kernels

$$\begin{array}{|ccc|} \hline -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \\ \hline \end{array} \quad \text{and} \quad \begin{array}{|ccc|} \hline -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \\ \hline \end{array} \tag{5.41}$$

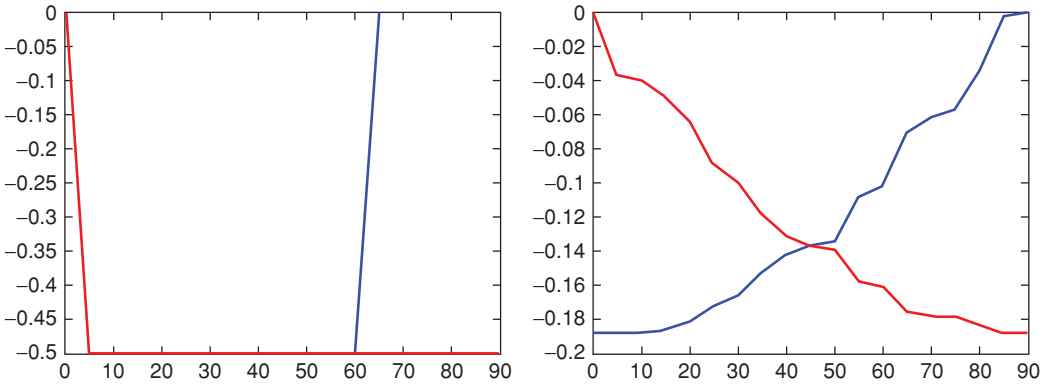


Figure 5.19 Results of using the simple 1×3 operators to estimate $\partial f/\partial x$ (in red) and $\partial f/\partial y$ (in blue). The horizontal axis is the orientation of the edge. The scale of the vertical axis is unimportant because the purpose of the experiment is to illustrate how poorly quantized the edge detectors are. LEFT: the two operators applied to perfect step edges. RIGHT: the two operators applied to blurred edges.

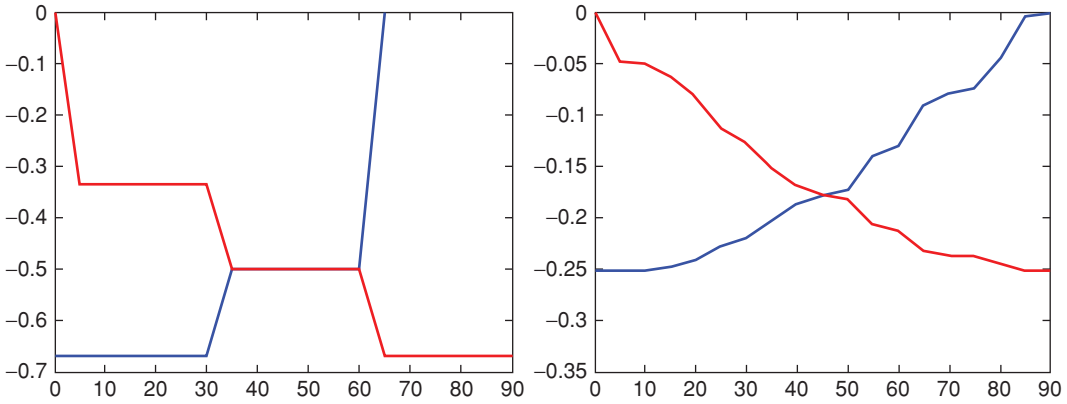


Figure 5.20 Results of applying the Sobel 3×3 operators to estimate $\partial f/\partial x$ (in red) and $\partial f/\partial y$ (in blue). The horizontal axis is the angle of the edge. LEFT: the two operators applied to perfect step edges. RIGHT: the two operators applied to blurred edges.

smooth the data by differentiating and summing, and also apply a center weight. Figure 5.20 illustrates the accuracy resulting from derivative estimates using the Sobel operators shown above. Here, a substantial improvement is observed in accuracy over the 1×3, but it is still not particularly good. Note that this heuristically motivated kernel looks a bit like a derivative of a Gaussian.

Optimal Biquadratic Fits

In this experiment, the minimum squared error methods of section 5.4.2 are used to determine the optimal fit of a plane to a surface. Taking the two first partial derivatives of the plane fit yield the two following derivative kernels for 5×5 kernels:

$$\begin{array}{|ccccc|} \hline -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ \hline \end{array} \quad \text{and} \quad \begin{array}{|ccccc|} \hline -2 & -2 & -2 & -2 & -2 \\ -1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ \hline \end{array} . \tag{5.42}$$

On the left, Figure 5.21 illustrates the result of applying this kernel to the ideal edges, and to the blurred images on the right. These kernels are non-intuitive, as they apply more weight to pixels further away from the point of interest. Indeed, these kernels perform poorly if the edges being detected are short or curved. The seemingly good performance demonstrated here is solely due to the fact that the edge being detected is long and straight, often an unrealistic assumption.

Derivative of a Gaussian

The two dimensional Gaussian,

$$h(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|K|^{1/2}} \exp\left(-\frac{[\mathbf{x}-\mu]^{\mathrm{T}} K^{-1} [\mathbf{x}-\mu]}{2}\right) \tag{5.43}$$

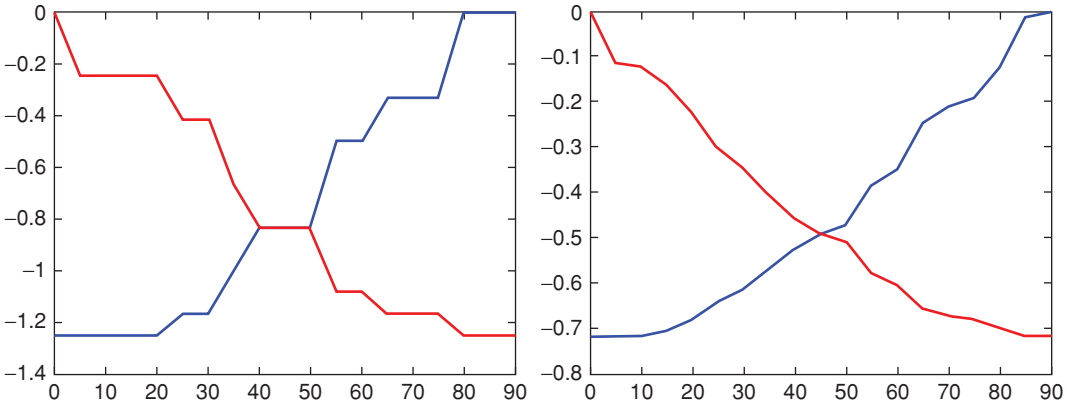


Figure 5.21 Results of applying the non-center-weighted operators of Eq. 5.42 to estimate $\partial f/\partial x$ (in red) and $\partial f/\partial y$ (in blue). The horizontal axis is the angle of the edge. The scale of the vertical axis is unimportant because the purpose of the experiment is to illustrate how poorly quantized the edge detetors are. LEFT: the two operators applied to perfect step edges. RIGHT: the two operators applied to blurred edges.

can be differentiated, creating

$$\frac{\partial h(x,y)}{\partial x} = \frac{-x}{2\pi\sigma^4}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{5.44}$$

for an isotropic covariance.

The derivative of a Gaussian (DoG) convolution kernels can be derived from the above equations by simply using the result of Eq. 5.44.

In Figure 5.22, the effect of scale, σ, is illustrated. As scale increases, the response of the operator to variations in orientation becomes increasingly smooth. This would suggest that the ability of the operator to determine orientation likewise depends on scale. That is true, but the scale of the features in the image must also be considered. This will be discussed in more detail in Chapter 11.

The Gabor Filter

Figure 5.23 illustrates the vertical edge detector, $\frac{\partial f}{\partial x}$, estimated using the Gabor operator. Similarity with the derivative of a Gaussian is observed when applied to the same set of edges with the same scale. However, the differences are significant – the response is not nearly as smooth. This would likely cause the Gabor to be less useful as an estimator of orientation.

5.8.2 Estimating Orientation

In this section, the ability of these operators to determine orientation is evaluated using the following algorithm, still applied to perfect and blurred step edges, respectively.

1. Apply a vertical and a horizontal version of the operator to each edge image, producing a set of estimates of gradients.
2. Using these estimates, apply Eq. 5.34 to estimate the orientation of the edge.
3. Plot the estimated angle vs. the actual angle of the edge.

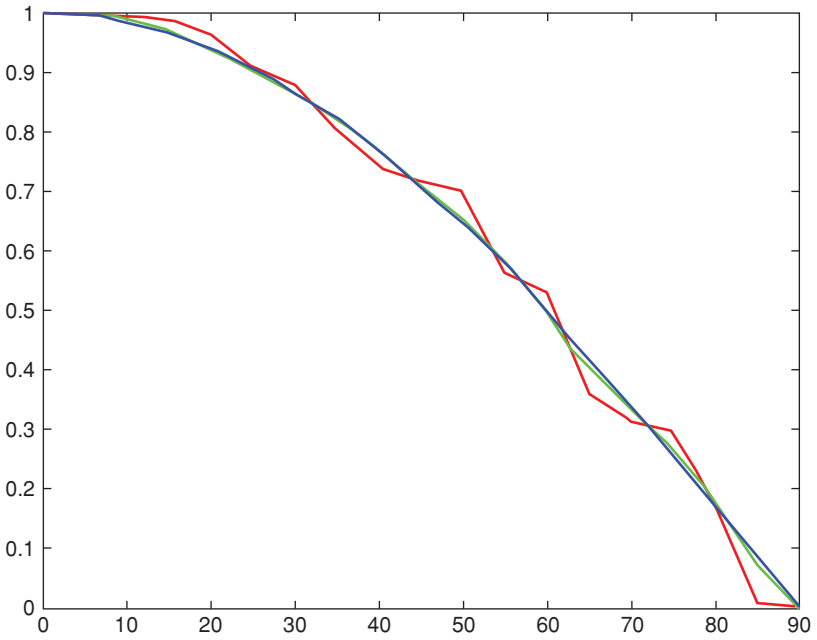


Figure 5.22 Results of applying derivative of Gaussian operators to estimate $\partial f/\partial x$, Eq. 5.42 with scales of 2 (red), 4 (green) and 6 (blue). The horizontal axis is the orientation of the edge.

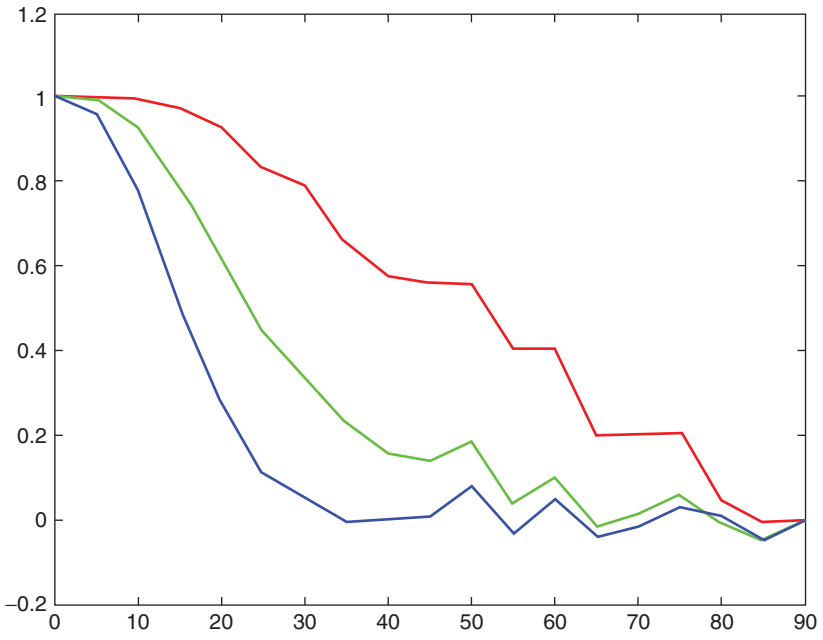


Figure 5.23 Results of applying a Gabor filter to estimate $\partial f/\partial x$ at a scale of 2 (in red), 4 (in green) and 6 (in blue) The horizontal axis is the angle of the edge.

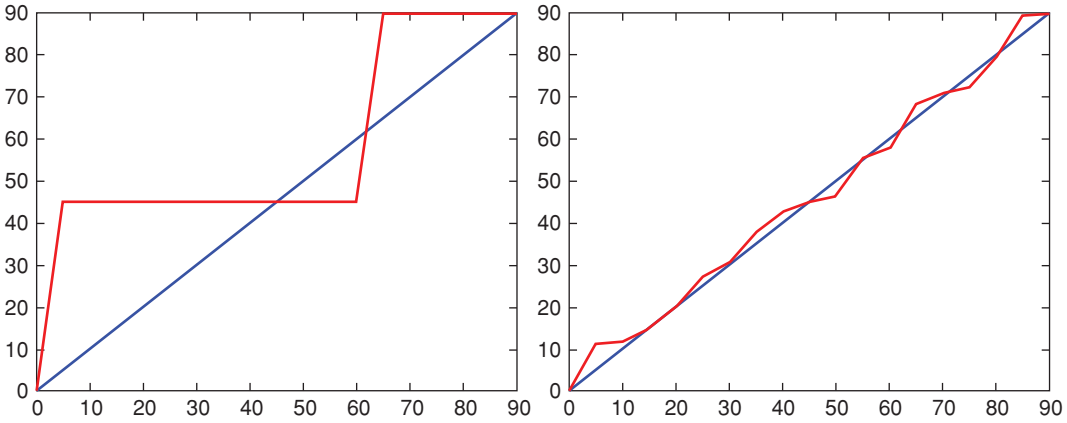


Figure 5.24 Estimate of edge orientation vs. actual orientation for 3×1 kernels used to estimate vertical and horizontal partial derivatives. The true edge orientation is in blue, and the estimated is in red. The left figure is the result for ideal edges and the right figure is that for blurred edges.

Simple 1×3 Filters

Figure 5.24 graphs the actual angle vs. the estimate, using 1×3 operators to estimate the directional derivatives. Clearly, the 1×3 operators are not useful for this application, at least not for perfect edges. As we have seen before, the slight blur spreads the information out and allows more accuate edge estimations.

Sobel Kernels

Figure 5.25 illustrates the accuracy resulting from derivative estimates using the Sobel operators shown previously, combined with Eq. 5.34. Here, a substantial improvement in accuracy is observed over the 1×3, but it is still not particularly good.

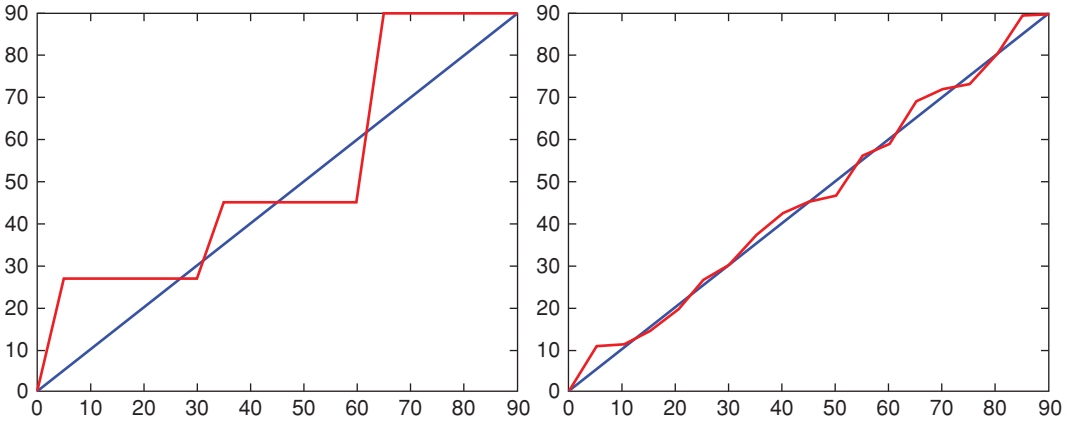


Figure 5.25 Using the two 3×3 Sobel kernels to estimate the angle produces somewhat better estimates, but still not very good. The edge used to produce the results in the left image was an ideal step edge, in the right image, a blurred step edge.

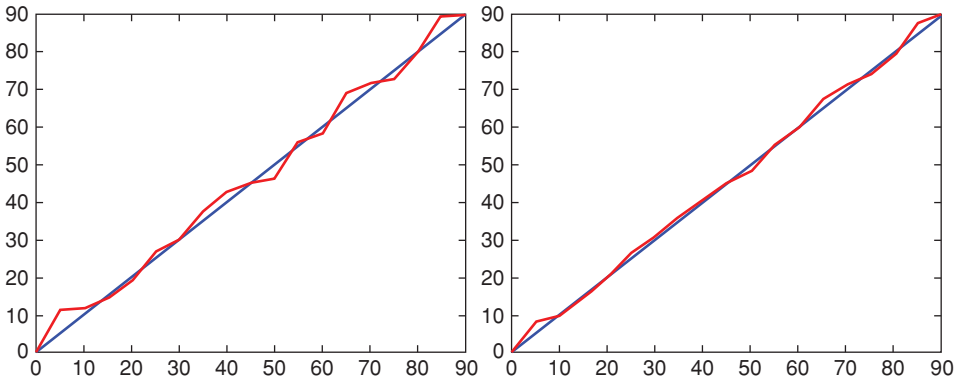


Figure 5.26 Estimate of edge orientation vs. actual orientation for kernels derived using derivatives of a Gaussian and used to estimate vertical and horizontal partial derivatives. The scale used was $\sigma = 2$.

Derivative of a Gaussian

Figures 5.26–5.28 graph the actual angle vs. the estimate, using three different operators that vary only in scale to estimate the directional derivatives. Clearly, the estimates are much better as the scale increases. This is due to the increased amount of blurring in the larger values of σ.

The Gabor Filter

Using two Gabor filters, one tuned to 90^o and the other to 0^o, we estimate the gradient. Like the derivative of a Gaussian, the Gabor filter contains an adjustable scale parameter, again called σ. In Figures 5.29–5.31, performance of Gabor filters having $\sigma = 2, 4, 6$ are illustrated.

The Gabor is much more selective to orientation than any of the other filters. That is, the Gabor produces a strong output near the orientation to which they are sensitive, and

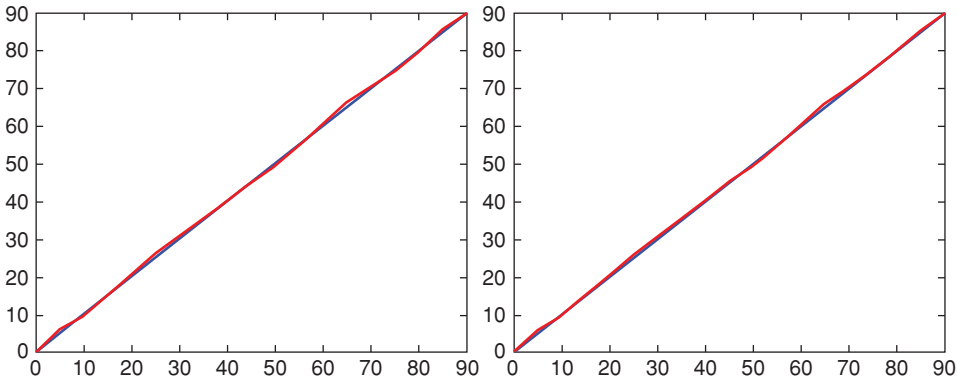


Figure 5.27 Estimate of edge orientation vs. actual orientation for kernels derived using derivatives of a Gaussian and used to estimate vertical and horizontal partial derivatives. The scale used was $\sigma = 4$.

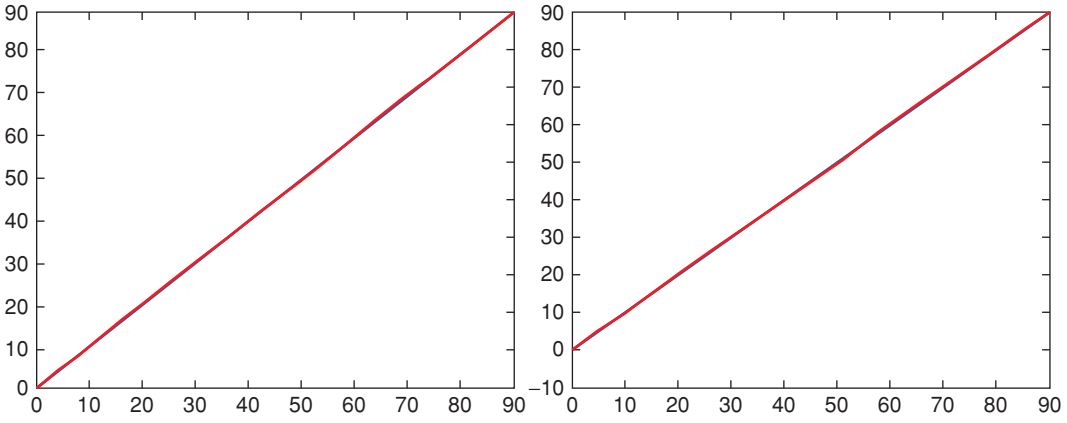


Figure 5.28 Estimate of edge orientation vs. actual orientation for kernels derived using derivatives of a Gaussian and used to estimate vertical and horizontal partial derivatives. The scale used was $\sigma = 6$.

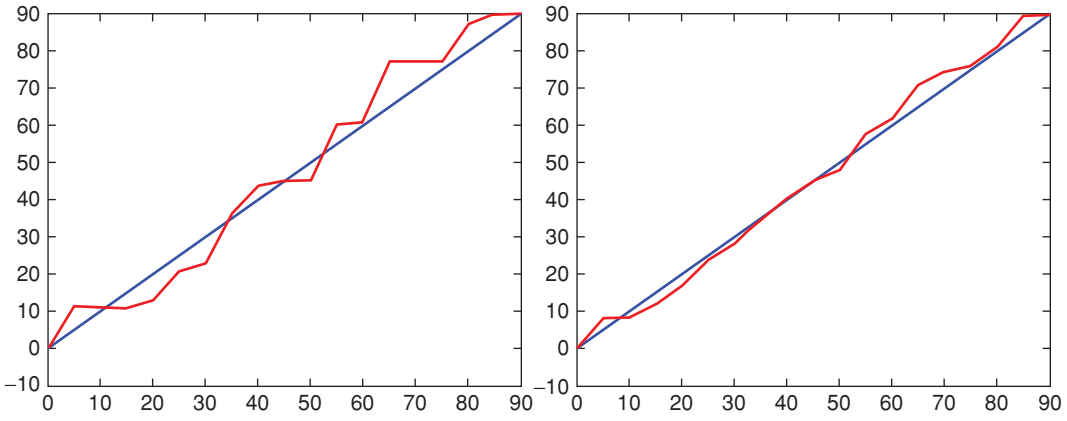


Figure 5.29 Estimates of edge orientation determined by applying a vertical and a horizontal Gabor filter. A scale of 2 is used.

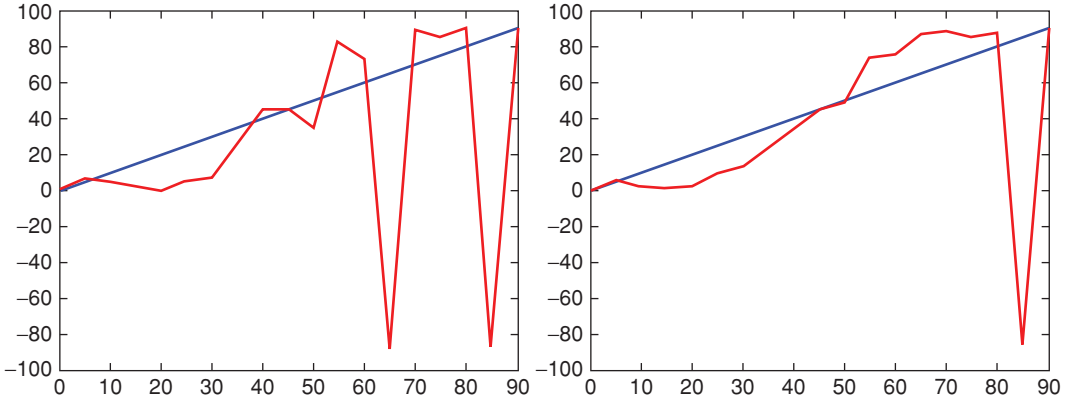


Figure 5.30 Estimates of edge orientation determined by applying a vertical and a horizontal Gabor filter. A scale of 4 is used.

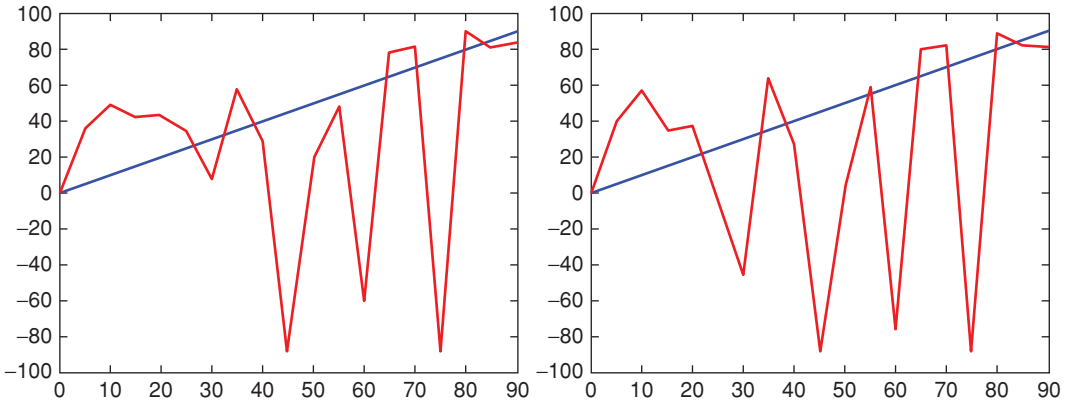


Figure 5.31 Estimates of edge orientation determined by applying a vertical and a horizontal Gabor filter. A scale of 6 is used.

essentially noise elsewhere. This is clearly seen in Figures 5.29–5.31. In the case of the Gabor filters used in these experiments, for most angles away from 0 or 90 degrees, one of the two measurements is meaningless; or worse, the second cycle of the sine wave may appear in the output, introducing a periodic distortion. With a larger σ, the exponential allows more cycles of the sine wave to affect the output, producing the surprising result that increasing σ decreases performance.

5.8.3 Discussion

The five functions discussed in this section, 1×3, Sobel, 5×5, Derivative of a Gaussian, and the Gabor, have many similarities. They can all be used to estimate the two derivatives from which the edge angle is determined. The best performer is the derivative of a Gaussian, especially with large σ.

But scale is a problem. With the derivative of a Gaussian, Figure 5.28 demonstrates essentially perfect performance. But how big must that kernel be? The function flDoG uses a kernel that goes out 3σ in order to accurately determine the value. For a σ of 6, this produces a kernel that is 37×37!

Remember that in most Computer Vision applications you want to find local features. A feature such as an edge that is straight for 37 pixels is, in most applications, a very large feature. Such a large kernel is therefore useful only for very large scale images. If $\sigma = 2$, reasonably good performance is obtained using just a 5×5 kernel.

5.9 Conclusion

In this chapter, we have looked at several ways to derive kernel operators that, when applied to images, result in strong responses for types of edges.

- We applied the definition of the derivative.
- We fit an analytic function to a surface, by minimizing the sum squared error.
- We converted subimages into vectors, and projected those vectors onto special basis vectors that described edge-like characteristics.

- We made use of the linearity of kernel operators to interchange the role of blur and differentiation to construct kernels. We used the constrained optimization and Lagrange multipliers to solve this problem.

Edge detection using derivative kernels, although effective, still suffers from many potential problems. In later chapters, we will discuss some more advanced algorithms to further improve the performance of edge detection.

For example, after we have chosen the very best operators to estimate derivatives, the best thresholds and the best estimates of edge position, there might still remain a set of pixels, marked as probably part of an edge. If those points are adjacent, one could "walk" from one pixel to the next, eventually circumnavigating a region. However, the points are unlikely to be connected the way we would like them to be. Some points may be missing due to blur, noise, or partial occlusion. There are many ways to approach this problem, including parametric transforms, which will be discussed in detail later in this book (Chapter 9).

To take another example, apply a Gaussian low-pass filter and then search for a zero in the (second) derivative could accurately (to sub-pixel resolution) find the exact location of the edge, but only if the edge is straight [5.29]. If the edge is curved, errors are introduced. In addition, all the methods cited or described in this chapter perform signal processing in a direction normal to the edge [5.17]. In order to better locate the actual edge, in Chapter 8, we will discuss advanced algorithms, which are more effective in locating curved edges through optimization. This improves edge detection over simple thresholding of the derivative.

We will wrap up this chapter by raising yet another issue in edge detection. Suppose we wish to ask a question like "is there an edge at orientation θ at this point?" How might we construct a kernel that is specifically sensitive to edges at θ? A straightforward approach [5.12] is to construct a weighted sum of the two Gaussian first derivative kernels, G_x and G_y, using a weighting something like

$$G_\theta = G_x \cos\theta + G_y \sin\theta \tag{5.45}$$

Could you calculate that orientation selectivity? What is the smallest angular difference you could detect with a 3×3 kernel determined in this way? However, unless quite large kernels are used, the kernels obtained in this way have rather poor orientation selectivity. In the event that we wish to differentiate across scale, the problem is even worse, since a scale-space representation is normally computed rather coarsely, to minimize computation time. Do you have a solution? If interested, we recommend the work by Perona [5.27] that provides an approach to solving these problems.

The authors are grateful to Bilge Karaçali and Rajeev Ramanath for their assistance in producing the images used in this chapter.

5.10 Assignments

Assignment 5.1: Section 5.4.2 showed how to estimate the first derivative by fitting a plane. Clearly that will not work for the second derivative, since the second derivative of a

plane is zero everywhere. Use the same approach, but use a biquadratic.

$$f(x, y) = ax^2 + by^2 + cx + dy + e$$

Then $A = [a\ b\ c\ d\ e]^{\mathrm{T}}$ and $X = [x^2\ \ y^2\ \ x\ \ y\ \ 1]^{\mathrm{T}}$.

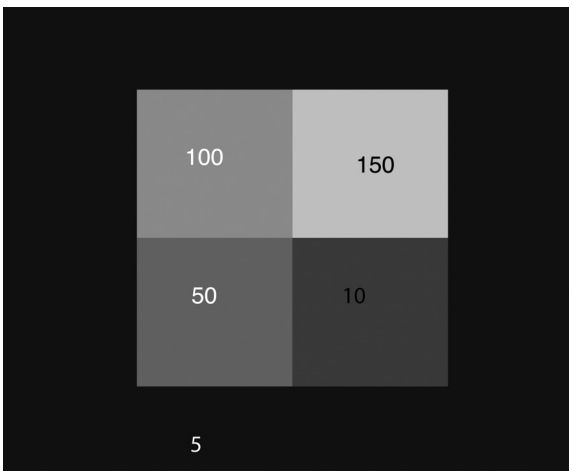
Now find the 3×3 kernel that estimates $\frac{\partial^2 f}{\partial x^2}$. Use a 3×3 neighborhood.

Assignment 5.2: Using the same methods as in section 5.4.2, instead of finding the 3×3 kernel, find the 5×5 kernel within a 5×5 neighborhood, which estimates $\frac{\partial f}{\partial x}$ at the center point, using the equation of a plane.

Assignment 5.3: Using the same methods as Assignment 5.1, but find the 5×5 kernel that estimates the second derivative at the center point, using the equation of a biquadratic.

Assignment 5.4: This assignment has several steps:

1. Create images:
   (a) Write a program to generate an image which is 64×64, as illustrated below



   The image should contain areas of uniform brightness as shown. Save this in a file and call it `SYNTH1.ifs`.

   (b) Write a program that reads in `SYNTH1.ifs`, and applies the blurring kernel

$$1/10 \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 2 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

   and write the answer to file named `BLUR1.ifs`

   (c) Add Gaussian random noise of variance $\sigma^2 = 9$ to `BLUR1.ifs`, and write the output to a file `BLUR1.V1.ifs`.

2. Implement edge detectors
   (a) Write a program to apply the following two kernels (referred to in the literature as the "Sobel operators") to images `SYNTH1.ifs`, `BLUR1.ifs`, and `BLUR1.V1.ifs`.

$$h_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad h_y = \begin{array}{|c|c|c|}\hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

   To accomplish this, perform the following;
   i. Apply h_x (see above) to the input, save the result as a temporary image in memory (remember that the numbers CAN be negative).
   ii. Apply h_y to the input, save the result as another array in memory.
   iii. Compute a third array in which each point in the array is the sum of the squares of the corresponding points in the two arrays you just saved. Finally, take the square root of each point. Save the result.
   iv. Examine the values you get. Presumably, high values are indicative of edges. Choose a threshold value and compute a new image, which is one whenever the edge strength exceeds your threshold and is zero otherwise.
   v. Apply steps 1–4 to the blurred and noisy images as well.
   (b) Write a report. Include a printout of all three binary output images. Are any edge points lost? Are any points artificially created? Are any edges too thick? Discuss sensitivity of the result to noise, blur, and choice of threshold. Be thorough; this is a research course that requires creativity and exploring new ideas, as well as correctly doing the minimum required by the assignment.

Assignment 5.5: In Assignment 5.2, you derived a 5×5 kernel that estimates $\frac{\partial f}{\partial x}$. Repeat Assignment 5.4 with the same images created, using that kernel for $\frac{\partial f}{\partial x}$ and the appropriate version for $\frac{\partial f}{\partial y}$.

Assignment 5.6: Verify the mathematics we did in Eq. 5.21. Those equations only show the x derivative for a one-dimensional Gaussian. To get the derivative to work correctly, it should differentiate in one direction and blur in the other. This requires the two-dimensional form of the Gaussian. Find a 3×3 kernel that implements the derivative-of-Gaussian vertical edge operator of Eq. 5.21. Use $\sigma = 1$ and $\sigma = 2$ and determine two kernels. Repeat for a 5×5 kernel. Discuss the impact of the choice of σ and its relation to kernel size. Assume the kernel may contain real (floating point) numbers.

Suppose the kernel can only contain integers. Develop kernels that produce approximately the same result.

Assignment 5.7: In section 5.4.4, parameters useful for developing discrete Gaussian kernels were discussed. Prove that the value of σ such that the maximum of the second derivative occurs at $x = -1$ and $x = 1$ is $\sigma = 1/\sqrt{3}$.

Assignment 5.8: Use the method of fitting a polynomial to estimate $\frac{\partial^2 f}{\partial y^2}$. Which of the following polynomials would be most appropriate to choose?

(a) $f = ax^2 + by + cxy$ (b) $f = ax^3 + by^3 + cxy$
(c) $f = ax^2 + by^2 + cxy + d$ (d) $f = ax + by + c$

Assignment 5.9: Fit the following expression to pixel data in a 3×3 neighborhood: $f(x, y) = ax^2 + bx + cy + d$. From this fit, determine a kernel that will estimate the second derivative with respect to x.

Assignment 5.10: Use the function $f = ax^2 + by^2 + cxy$ to find a 3×3 kernel that estimates $\frac{\partial^2 f}{\partial y^2}$. Which of the following is the kernel that results? (Note: the following answers do not include the scale factor. Thus, the best choice below will be the one proportional to the correct answer.)

(a)
6	4	0
4	6	0
0	0	4

(b)
2	6	2
−4	0	−4
2	6	2

(c)
6	4	0
4	6	0
0	0	1

(d)
1	−2	1
3	0	3
1	−2	1

(e)
1	0	−1
−2	0	2
1	0	−1

(f)
1	2	1
−2	0	−2
−1	−2	−1

Assignment 5.11: What does the following expression estimate?

$$(f \otimes h_1)^2 + (f \otimes h_2)^2 = E$$

where the kernel h_1 is defined by

0.05	0.08	0.05
−0.136	−0.225	−0.136
0.05	0.08	0.05

,

and h_2 is created by the transpose. Choose the best answer
a) The first derivative with respect to x (where x is the horizontal direction)
b) The second derivative with respect to x
c) The Laplacian
d) The second derivative with respect to y
e) The quadratic variation

Assignment 5.12: You are to use the idea of differentiating a Gaussian to derive a kernel. What variance does a one-dimensional (zero mean) Gaussian need to have to have the property that the extrema of its first derivative occur at $x = \pm 1$?

Assignment 5.13: What is the advantage of the quadratic variation as compared to the Laplacian?

Assignment 5.14: Let $E = (\mathbf{f} - H\mathbf{g})^{\mathrm{T}}(\mathbf{f} - H\mathbf{g})$. Using the equivalence between the kernel form of a linear operator and the matrix form, write an expression of E using kernel notation.

Assignment 5.15: The purpose of this assignment is to walk you through the construction of an image pyramid, so that you will more fully understand the potential utility of this data structure, and can use it in a coding and transmission application. Along the way, you will pick up some additional understanding of the general area of image coding. Coding is not a principal learning objective of this course, but in your research career, you are sure

to encounter people who deal with coding all the time. It will prove to be useful to have a grasp of some of the more basic concepts.

1. Locate the image specified by your instructor. Let's call that image XXX512. Verify that it is indeed 512×512. If it is not 512×512, then first write a program that will pad it out to that size.
2. Write a program that will take a two-dimensional $n \times n$ image and produce an image, of the same data type, which is $n/2 \times n/2$. The program name should be `ShrinkByTwo` and called by

   ```
   ShrinkByTwo inimg outimg
   ```

   The program should *not* simply take every other pixel. Instead, each pixel of the output image should be constructed by averaging the corresponding four pixels in the input image. Note the requirement that the output image be of the same type as the input. Use your program to create XXX256, XXX128, XXX64, and XXX32.

   Don't bother with going below a 32×32 image. Turn in your program and a print of your images.
3. Write a subroutine to zoom an image. It should have the following calling convention:

   ```
   ZoomByTwo(inimg,outimg)
   ```

   The calling program is responsible for creating, reading, etc. of the images. The subroutine simply fills `outimg` with a zoomed version of `inimg`. Use any algorithm you wish to fill in the missing pixels. (We recommend that the missing pixels be some average of the input pixels).

   Before we can proceed, we need to consider a "pyramid coder." When you ran `ShrinkByTwo`, the set of images you produced is the pyramid representation of `XXX512`. In a pyramid coder, the objective is to use the pyramid representation to transmit as little information as possible over the channel. Here is the idea: First, transmit all of `XXX32`. Then, both the transmitter and receiver run `ZoomByTwo` to create a zoomed version of `XXX32`, something like

   ```
   ZoomByTwo(XXX32,a64prime)
   ```

   When we created `XXX32` from `XXX64`, we threw away some information, and we cannot easily get it back, so `a64prime` will not be identical to `XXX64`. If, however, `ZoomByTwo` (which in the image coding literature, is called the "predictor") is pretty good, the difference between `a64prime` and `XXX64` will be small (small in value that is, it is still 64×64). Therefore, compute `diff64`, the difference between `a64prime` and `XXX64`. If the predictor were perfect, the resultant difference would be a 64×64 image of all zeros, which could be coded in some clever way (by run-length encoding for example), and transmitted with very few bits. Let us now transmit `diff64` to the receiver. By simply adding `diff64` to the version of `a64prime` generated at the receiver, we can correct the errors made by the predictor, and we now have a correct version of `XXX64` at the receiver, but all we transmitted was `diff64`. Clever, aren't we?

Now, from `XXX64`, we play the same game and get `XXX128` created by transmitting `diff128`, et cetera, ad nauseam. Now, the assignment:

4. Create the images described above, `diff64`, `diff128`, `diff256`, `diff512`. Measure the approximate number of bits required to transmit each of them. To make that measurement: Compute the standard deviation of, say, `diff64`. Take the log base 2 of that standard deviation, and that is the average number of bits per pixel required to code that image. Assume you were to transmit `XXX512` directly. That would require $512 \times 512 \times 8$ bits (assuming the image is 8 bits; you better verify). Now, compare with the performance of your pyramid coder by adding up all the bits/pixel you found for each of the diff images you transmitted. Did your coder work well? Discuss this in your report.

Assignment 5.16: At high levels of scale, only _____ object are visible. (fill in the blank)

Bibliography

[5.1] M. Alfaraj, Y. Wang, and Y. Luo. Enhanced isotropic gradient operator. *Geophysical Prospecting*, 62, 2014.

[5.2] V. Anh, J. Shi, and H. Tsai. Scaling theorems for zero crossings of bandlimited signals. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(3), 1996.

[5.3] J. Babaud, A. Witkin, M. Baudin, and R. Duda. Uniqueness of the Gaussian kernel for scale-space filtering. *IEEE Trans. Pattern Anal. and Machine Intel.*, 8(1), 1986.

[5.4] G. Bilbro and W. Snyder. Optimization of functions with many minima. *IEEE Transactions on SMC*, 21(4), July/August 1991.

[5.5] P. Burt and E. Adelson. The Laplacian pyramid as a compact image code. *CVGIP*, 16, 1981.

[5.6] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. and Machine Intel.*, 8(6), 1986.

[5.7] J. Crowley. *A Representation for Visual Information*, Ph.D. Thesis. CMU, 1981.

[5.8] J. Daugman. Two-dimensional spectral analysis of cortical receptive fields. *Vision Research*, 20, 1980.

[5.9] J. Daugman. Uncertainly relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *J. Optical Soc. America*, 2(7), 1985.

[5.10] W. Frei and C. Chen. Fast boundary detection: A generalization and a new algorithm. *IEEE Transactions on Computers*, 25(2), 1977.

[5.11] R. Haralick and L. Shapiro. *Computer and Robot Vision*, volume 1. Addison-Wesley, 1992.

[5.12] M. Van Horn, W. Snyder, and D. Herrington. A radial filtering scheme applied to intracoronary ultrasound images. *Computers in Cardiology*, Sept. 1993.

[5.13] D. Hubel and T. Wiesel. Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology (London)*, 160, 1962.

[5.14] D. Hubel and T. Wiesel. Functional architecture of macaque monkey visual cortex. *Proceedings of the Royal Society, B*, 198, 1978.

[5.15] P. Jackway and M. Deriche. Scale-space properties of the multiscale morphological dilation-erosion. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(1), 1996.

[5.16] J. Jones and L. Palmer. An evaluation of the two-dimensional gabor filter model of simple receptive fields in the cat striate cortex. *J. Neurophysiology*, 58, 1987.

[5.17] E. Joseph and T. Pavlidis. Bar code waveform recognition using peak locations. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(6), 1994.
[5.18] M. Kelly. *Machine Intelligence 6*. Univ. Edinburgh Press, 1971.
[5.19] M. Kisworo, S. Venkatesh, and G. West. Modeling edges at subpixel accuracy using the local energy approach. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(4), 1994.
[5.20] T. Lee. Image representation using 2-d Gabor wavelets. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(10), 1996.
[5.21] Y. Leung, J. Zhang, and Z. Xu. Clustering by scale-space filtering. *IEEE Trans. Pattern Anal. and Machine Intel.*, 22(12), 2000.
[5.22] T. Lindeberg. Scale-space for discrete signals. *IEEE Trans. Pattern Anal. and Machine Intel.*, 12(3), 1990.
[5.23] T. Lindeberg. Scale-space theory, a basic tool for analysing structures at different scales. *Journal of Applied Statistics*, 21(2), 1994.
[5.24] D. Marr and E. Hildreth. Theory of edge detection. *Proc. Royal Society of London, B*, 207, 1980.
[5.25] D. Marr and T. Poggio. A computational theory of human stereo vision. In *Proc. Royal Society of London*, 1979.
[5.26] E. Pauwels, L. Van Gool, P. Fiddelaers, and T. Moons. An extended class of scale-invariant and recursive scale space filters. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(7), 1995.
[5.27] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 12, 1990.
[5.28] H. Tagare and R. deFigueiredo. Reply to on the localization performance measure and optimal edge detection. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(1), 1994.
[5.29] P. Verbeek and L. van Vliet. On the location error of curved edges in low-pass filtered 2-d and 3-d images. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(7), 1994.
[5.30] I. Weiss. High-order differentiation filters that work. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(7), 1994.
[5.31] R. Young. The Gaussian derivative model for spatial vision: I. retinal mechanisms. *Spatial Vision*, 2, 1987.

6 Noise Removal

To change and to change for the better are two different things. – German proverb.

6.1 Introduction

In a photosensitive device such as a phototransistor or charge-coupled device, an incidence of a photon of light may (probabilistically) generate an electronic charge. The number of charges produced should be proportional to the photons per second striking the device. However, the presence of heat (anything above absolute zero) will also randomly produce charges, and therefore signal. Such a signal is called *dark current* because it is a signal that is produced by a camera, even in the dark. Dark current is one of several phenomena that result in random fluctuations to the output of a camera that we call *noise*. The nature of noise is closely related to the type of sensor. For example, devices that count emissions of radioactive particles are corrupted by a noise that has a Poisson distribution rather than the Gaussian noise of dark current.

In this chapter, techniques are developed that remove noise and degradations so that features can be derived more cleanly for segmentation. We will introduce each topic in one dimension, to allow the student to better understand the process, and then extend that concept to two dimensions. This is covered in the following sections:

- (Section 6.2) The noise in the image can be reduced simply by smoothing. However, the smoothing process also blurs the edges. This section introduces the subject of reducing the noise while at the same time preserving edges, i.e., edge-preserving smoothing.
- (Section 6.3) An intuitive idea in designing edge-preserving smoothing is that smoothing should be associated with a weight according to the local image data where it is applied. And the weight should be large if two pixels are close spatially and have similar photometric values. Otherwise, the weight should be small. The bilateral filter is an algorithm that realizes this ad hoc "good idea."
- (Section 6.4) Diffusion is described here to pose the denoising problem as the solution to a partial differential equation (PDE). The challenge is how to find a PDE that causes blurring except at edges.
- (Section 6.5) The Maximum *A Posteriori* probability (MAP) algorithm is discussed to show how to formulate noise removal as a minimization problem. Here, the challenge is to find an objective function whose minimum is the desired result.

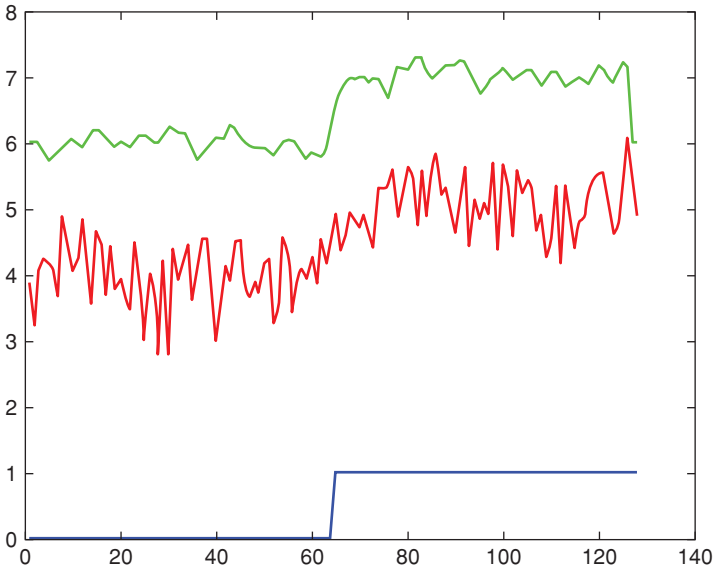


Figure 6.1 (Blue) A graph of a signal with a perfect step edge. (Red) The same signal with Gaussian random noise added. (Green) The signal resulting from blurring with the kernel in Eq. 6.1.

The diffusion-based approach, e.g., Variable Conductance Diffusion (VCD), is compared with the MAP-based optimization method, and the two methods are shown to be equivalent. See section 6.6 for detail. The equivalence shows the relationship between energy minimization methods and spatial analysis methods and between their respective parameters of temperature and scale.

6.2 Image Smoothing

We begin by introducing a common concept, image smoothing. Initially, we discuss it in the context of a simple one-dimensional function.

6.2.1 The 1D Case

The noise in a one-dimensional discrete signal $f(x)$ can be reduced simply by blurring the signal using a simple kernel like

1	2	4	2	1

(6.1)

which is a five-point approximated version of the one-dimensional Gaussian (recall section 5.4.4). In Figure 6.1, a step edge (recall section 5.5) with heights of zero and one has Gaussian random noise added, and then is blurred to remove the noise. Note that the blurred signal has reduced noise, but the edge is no longer sharp.

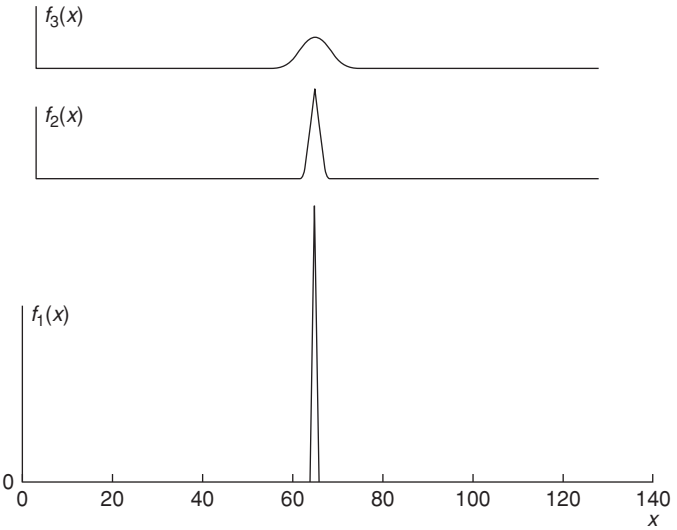


Figure 6.2 Three functions of x: an impulse at $x = 64$, and two versions of convolution of this impulse with Gaussians of $\sigma = 1$ (lower), and $\sigma = 3$ (upper).

6.2.2 The 2D Case

In two dimensions, we use a two-dimensional Gaussian approximation like

1	2	1
2	4	2
1	2	1

.

Such a kernel simply blurs the image. Indeed, that will reduce noise in the image, but it will also blur the edges and seriously degrade the image quality. Therefore, we will need to find a method that reduces the noise, but still results in sharp edges. Such an algorithm is called *edge-preserving smoothing*. However, before we think about edge-preserving smoothing, we need to understand simply smoothing first.

Convolution[1] with a Gaussian blurs an image very effectively, but raises the question "what is the best choice of σ?" For convenience, we repeat the equation for a 2D Gaussian, Eq. 5.19:

$$h(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{[x\ \ y][x\ \ y]^{\mathrm{T}}}{2\sigma^2}\right) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{6.2}$$

To get more blur, just make σ larger. But, making σ larger will require a larger kernel, since the kernel size is usually chosen to be 2 or 3 σ, which in turn causes additional problems at image boundaries and requires more compute time.

In Figure 6.2, a one-dimensional impulse is illustrated, as well as two blurs of that impulse. Recall that all three curves integrate to the same value.

[1] Note that in this chapter, we use the term "convolution" to actually indicate the "sum-of-product" operator. Since the Gaussian kernel is symmetric, convolution here is actually equivalent to correlation.

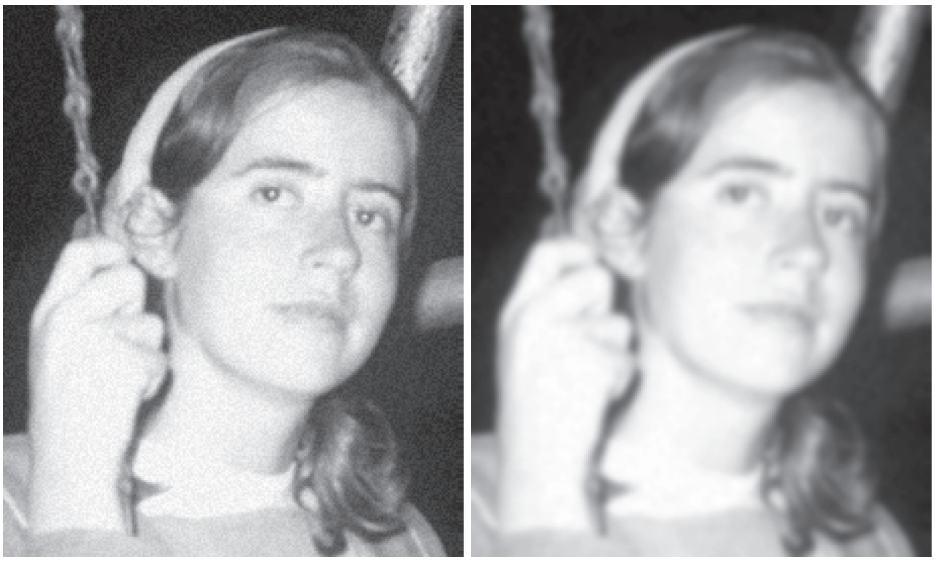


Figure 6.3 LEFT: A noisy grayscale image (Library name: swinggray40.png). RIGHT: The same image after Gaussian blur.

So, let's use small kernels. If we convolve twice, using Gaussians of width σ_1 and σ_2, this is equivalent to convolving with a kernel of $\sigma = \sqrt{\sigma_1^2 + \sigma_2^2}$. Therefore, smoothing can be accomplished by repeated application of convolution with a small kernel. The use of small kernels allows the blurring to approach the boundaries of the image much more closely.

In Figure 6.3, a noisy version of a grayscale image is shown, and in Figure 6.4, a zoom of a segment of the same image. The zoom allows us to more effectively evaluate the effectiveness of blur removal.

The blurring process has removed the noise, but also seriously damaged fine detail. From this result, it is apparent that a method is needed that can blur out the noise while preserving

Figure 6.4 LEFT: A noisy grayscale image, the zoom of the previous image (Library name: swingeyegray40.png). RIGHT: The same image after Gaussian blur.

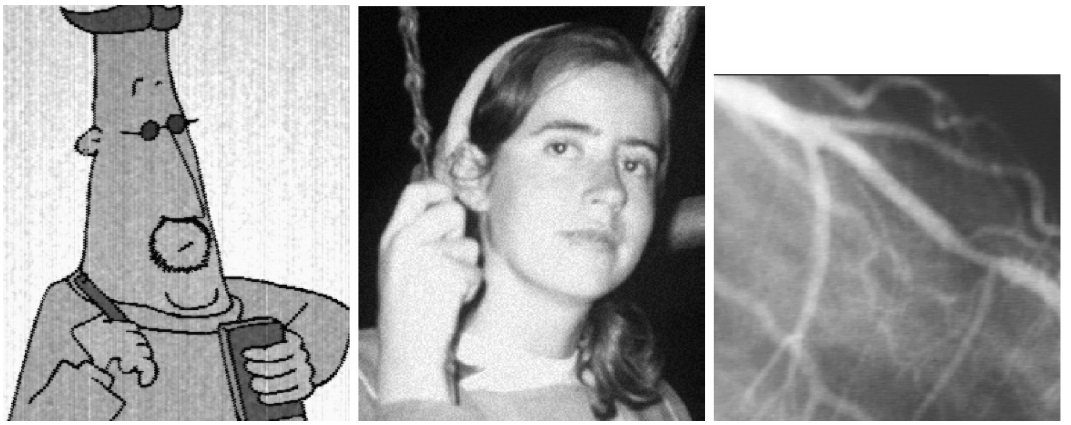


Figure 6.5 (left-to-right) A cartoon scanned by a grayscale camera. (Caraman/iStock/Getty Images Plus, used with permission). (Library name: cartoon1.png), a noisy grayscale image (Library name: swinggray40.png), an angiogram (Library name: angiof.png).

the edges. In the following sections, three methods will be described. The three methods will be used to both illustrate the method itself, and the underlying relevant mathematical techniques.

Test Images

In the subsections to follow, each of these methods is demonstrated as it operates on the three images shown in Figure 6.5.

The first test image is a girl in a swing, Figure 6.3. This image has some sharp edges at the boundaries between face, hair, and other features, but over the face, the brightness changes gradually except for noise.

The second is an *angiogram*, which is an x-ray of the coronary arteries of a living heart. In this case, in addition to the Poisson instrument noise, the image contains "clutter." Clutter objects in an image are not of interest, but are real. In this image the clutter results from the presence of many other blood vessels around the large coronary artery in the center of the image.

The third is an image of a cartoon, scanned with a poor-quality camera with a poor lens and a defective analog-to-digital converter.

In Figure 6.6 we illustrate one way to visualize the amount of high-frequency noise in an image. A random color has been assigned to each brightness level. Because the colors are random, two pixels may have similar brightnesses but be assigned very different colors. Only if two pixels have identical brightnesses will they have the same color. Good noise removal tends to produce patches that have identical colors. Figure 6.6 provides the color maps of these images. The random nature of all three is quite apparent.

6.3 Edge-Preserving Smoothing Using Bilateral Filters

In this section you will learn how a bilateral filter can be used to smooth an image while preserving edges. A bilateral filter simply convolves the image with a kernel, as was done

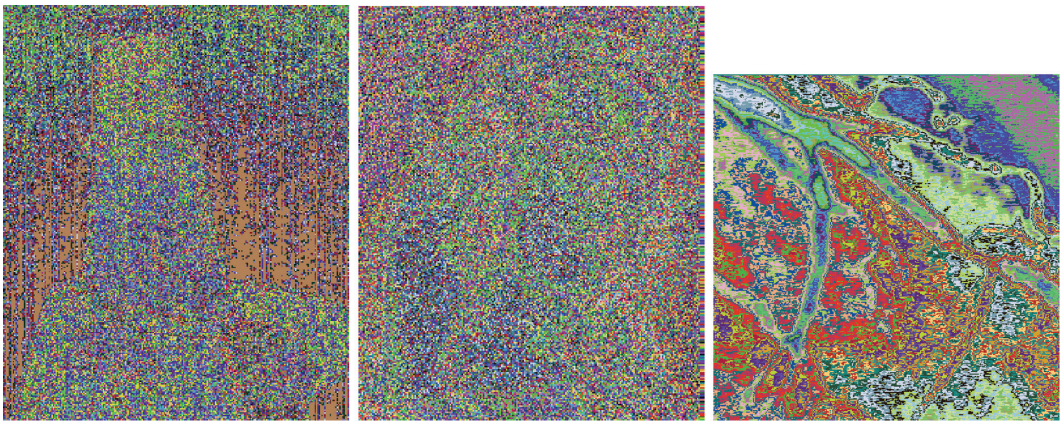


Figure 6.6 (left-to-right) Using a random color map to show noise: A cartoon scanned by a grayscale camera, a noisy grayscale image, and an angiogram.

many times in previous chapters. The difference here is that the weights of the filter vary according to the image data where they are applied. You may have encountered similar systems in your signal processing courses, where you encountered "linear, time-invariant systems," and learned about the Fourier transform. Here, in the signal processing analogy, this would be a time-variant system. Of course, in image processing, we think of space-variant rather than time-variant, but the mathematics is the same.

The idea is this: at every point in the image, we compute a weighted sum of the neighbors of that point, just as we do with a regular kernel operator. However, if the neighbor is far away, we do not consider it very strongly, and similarly, if the brightness of the neighbor is very different from the center pixel, it is also not strongly considered. Paris [6.22] summarizes it nicely, saying "two pixels are close to each other not only if they occupy nearby spatial locations but also if they have some similarity in the photometric range."

This is accomplished at a point $\mathbf{x} = [x, y]^{\mathrm{T}}$ by computing a new value for $f(\mathbf{x})$ using

$$f(\mathbf{x}) = \sum_i f(\mathbf{x}_i) G_{\sigma_d}(||\mathbf{x} - \mathbf{x}_i||) G_{\sigma_f}(f(\mathbf{x}) - f(\mathbf{x}_i)) \,, \tag{6.3}$$

where $G_\sigma(t)$ is a 2D Gaussian with width parameter (standard deviation) σ, and

$$G_\sigma(t) = \frac{1}{2\pi\sigma} \exp\left(-\frac{t^2}{2\sigma^2}\right) \,.$$

The norm $||.||$ denotes the 2-norm, the Euclidean distance from point $\mathbf{x}$ to each of the neighboring points $\mathbf{x}_i$.

The product of the two Gaussians will be small if either is small, that is, if the point $\mathbf{x}_i$ is far from $\mathbf{x}$, or if the two points differ significantly in brightness. The two σ parameters control what is considered a reasonable distance in the image or difference in brightness for a particular application.

Equation 6.3 adds up weights that are data dependent, and therefore must be normalized for the result to make sense. Normalization proceeds by computing a normalization term

$$Z = \sum_i G_{\sigma_d}(||\mathbf{x} - \mathbf{x}_i||) G_{\sigma_f}(f(\mathbf{x}) - f(\mathbf{x}_i)) \,,$$

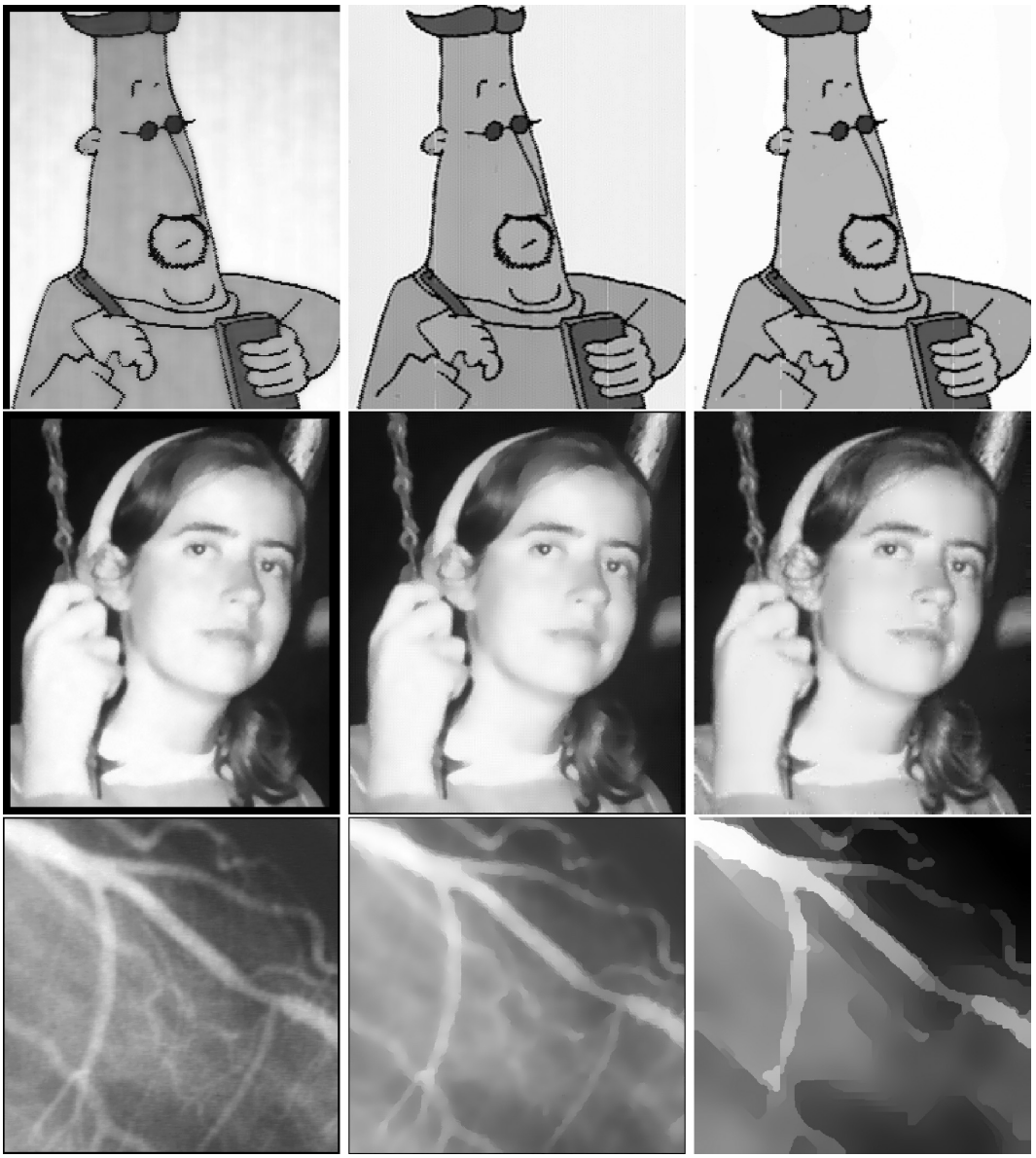


Figure 6.7 The first column represents the example images processed by a bilateral filter, the second column by VCD, and the third by an MAP method using a piecewise-constant prior.

and revising Eq. 6.3 to be

$$f(\mathbf{x}) = \frac{1}{Z} \sum_i f(\mathbf{x}_i) G_{\sigma_d}(||\mathbf{x} - \mathbf{x}_i||) G_{\sigma_f}(f(\mathbf{x}) - f(\mathbf{x}_i)) \,, \tag{6.4}$$

Comparing Figure 6.4 with the first column of Figure 6.7, we see that using the bilateral filter removes the noise and leaves the details sharper. In the center image, large areas of

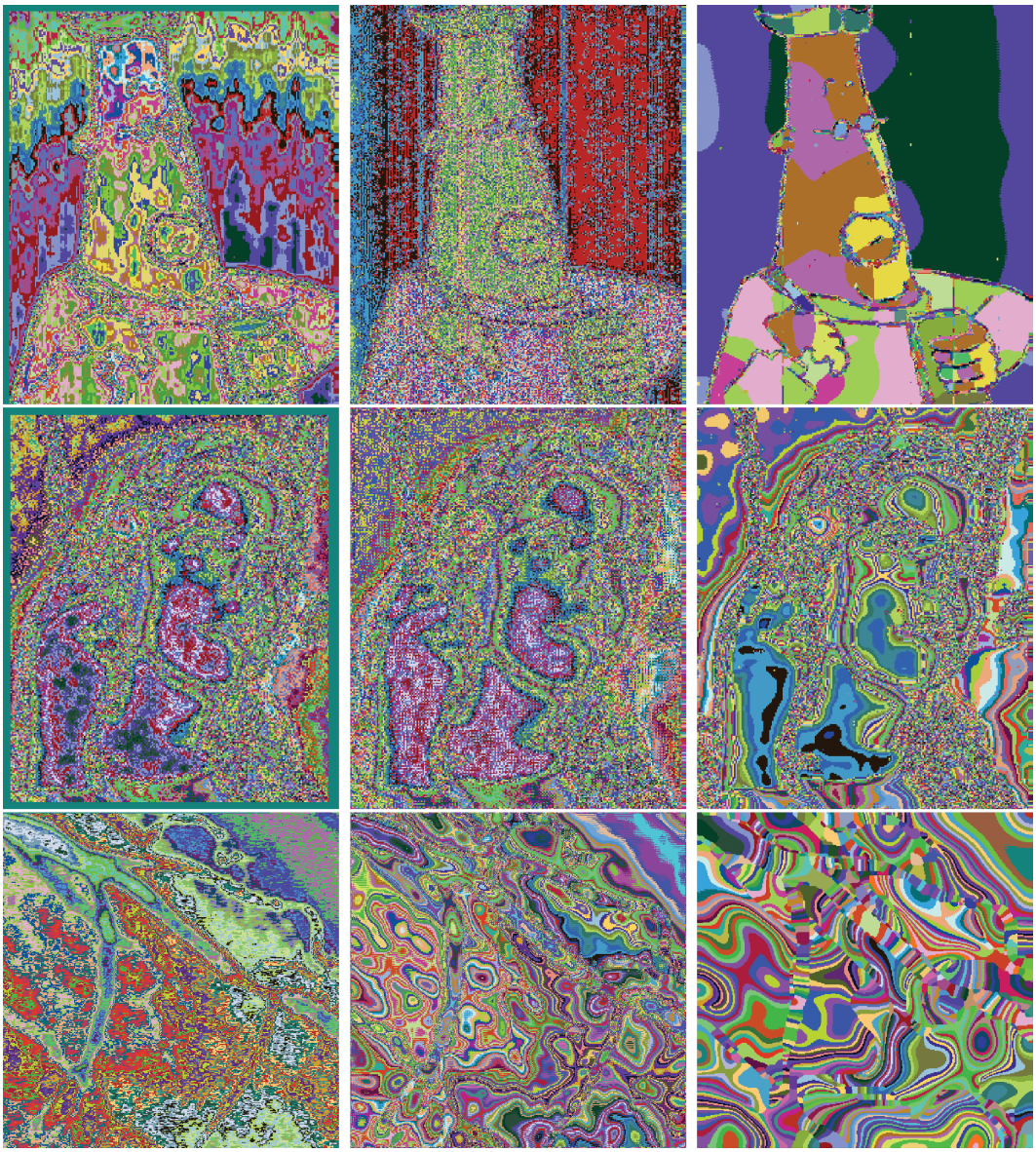


Figure 6.8 The images in the previous figure, illustrated using random color maps. The first column represents the example images processed by a bilateral filter, the second column by VCD, and the third by an MAP method using a piecewise-constant prior.

uniform brightness can be observed. This may be considered one of the weaknesses of bilateral filtering when applied to images with smoothly changing brightness.

The denoising effect is also illustrated in the first column of Figure 6.8 using the random color map.

In the next section, a different approach will be presented to solve the edge-preserving smoothing problem, and give you the opportunity to learn how the use of partial differential equations can be applied to Computer Vision problems.

6.4 Edge-Preserving Smoothing Using the Diffusion Equation

In this section you will learn the relationship between diffusion and blurring, what a Green's function is, and how to use variable-conductance diffusion to do edge-preserving smoothing of an image.

6.4.1 The Diffusion Equation in One Dimension

In one dimension, the "diffusion equation" or "heat equation" is the partial differential equation (PDE),

$$\frac{\partial f(x,t)}{\partial t} = \frac{\partial}{\partial x}\left[c\frac{\partial f(x,t)}{\partial x}\right] \tag{6.5}$$

where c is a scalar known as "conductance." This equation models the distribution of heat in a material over time.

Think about Eq. 6.5. Both sides are functions of time, and the left-hand side is the rate of change with respect to time. In that equation, we use the notation of differential calculus. Such notation is compact, general, and relatively easy to use when we need to understand the properties of systems or processes. Ultimately, however, we will implement them on a digital computer, which performs operations that are discrete in time and space. Therefore partial derivatives cannot be truly computed, but rather must be estimated, using some discrete estimator such as:

$$\frac{\partial f(x,t)}{\partial t} \approx f(x,t) - f(x,t-1)\ .$$

Given an initial function $f(x,t_0)$, some estimator, like the right-hand side of Eq. 6.5, can provide an estimate of the time change. Since that is the change with respect to time, it is added to the current estimate,

$$f(x,t_1) = f(x,t_0) + \Delta_t f(x,t_0)\ .$$

If a diffusion is run for a time, t, the function $f(x,t)$ becomes more and more blurred as t grows larger. This blurring will be exactly the same as what you would find if the starting image were convolved with a Gaussian kernel with a σ that is a particular function of t.

This relationship between running a PDE over time and the existence of a convolution that does the same thing is the definition of a *Green's function*. That is: Suppose we have a PDE that, when run on an image f with a particular set of initial and boundary conditions, produces a result g. If there exists a kernel that, when convolved with f, also produces g, we say the kernel is the "Green's function" of the PDE. In the case we are describing, convolution with a Gaussian produces the same result as running the diffusion equation (except, of course, for the difference between discrete convolution (a sum of products) and continuous convolution (an integral of a product). Therefore, we say the Gaussian is the Green's function of the diffusion equation.

Therefore, one can simulate the PDE or one can convolve. The simulation of the PDE as t goes from 0 to T is equivalent to convolution with a Gaussian with σ of

$$\sigma = \sqrt{2cT}\ . \tag{6.6}$$

6.4.2 Simulation of a PDE

How, then, does one simulate a PDE? For this PDE, which has the change with respect to time on the left, it's really easy. Just think of the equation in discrete time and write, for pixel i,

$$\frac{\Delta f_i(x,t)}{\Delta t} = \frac{\partial}{\partial x}\left[c\frac{\partial f(x,t)}{\partial x}\right]\Bigg|_i , \tag{6.7}$$

and then let Δt be the time it takes for one iteration through the program. You already know how to approximate the right-hand side. The left-hand side is the change at each iteration. So, as mentioned above, compute the right-hand side and add that to f in each iteration.

6.4.3 The Diffusion Equation in Two Dimensions

Equation 6.7 provides a way to continuously smooth a one-dimensional function. The previous description can be generalized to two dimensions by using the vector gradient as follows. (Remember, the vector gradient operator in two dimensions is $\nabla = \left[\frac{\partial}{\partial x}\ \frac{\partial}{\partial y}\right]^{\mathrm{T}}$).

$$\frac{\partial f_i}{\partial t} \approx \alpha(\nabla^{\mathrm{T}}(c\nabla f))\Big|_i , \tag{6.8}$$

which would be simulated in the same way. The multiplicative scalar α is a result of the fact that we aren't really using infinitesimal derivatives. Choose α to be a small positive constant, like 0.01 when you actually implement this. This will be discussed in more detail when we consider gradient descent.

6.4.4 Variable Conductance Diffusion (VCD)

The problem with simulating the diffusion equation is, of course, that it blurs everything. It takes out noise and it takes out edges. This problem can be fixed by considering the scalar c.

First, let c be a scalar function of position, $c(x, y)$.

To smooth, except at edges, let $c(x, y)$ be small if (x, y) is an edge pixel. If $c(x, y)$ is small (in the heat transfer analogy), little heat flows, and in the image, little smoothing occurs. If, on the other hand, $c(x, y)$ is large, then much smoothing is allowed in the vicinity of pixel (x, y). VCD then implements an operation that after repetition produces a nearly piecewise uniform result. It only smoothes between edges, thus preserving those edges.

Since c is a function of space, it cannot be factored out of Eq. 6.5 to produce a nice second derivative on the right. But what form should c take? It can be any function of space that is close to zero in areas of the image where the image is very "busy."[2] One possible form is:

$$c(x, y) = \exp\left(-\Lambda(f(x, y))\right) \tag{6.9}$$

where $\Lambda(f(x, y))$ is some measure of how strong the edges are in the vicinity of (x, y). The choice of a particular form for c is problem-dependent, but when the edge is strong, Λ should be large, so that the exponential is near zero, and there is no blurring. If there are no

[2] That is, the intensity values of the pixels vary significantly.

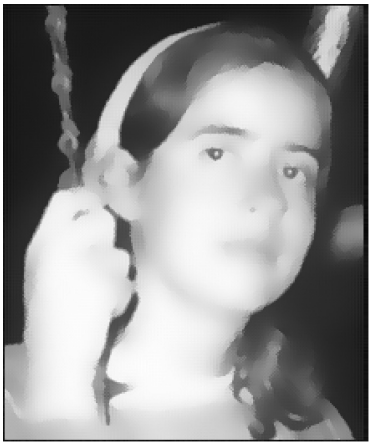


Figure 6.9 Running a simple diffusion too long can result in an image that still has sharp exterior edges but has lost interior detail.

edges, Λ should be small, the exponential is near one, and blurring proceeds. One example of $c(x, y)$ that provides a good compromise between performance and computation is

$$c(x, y) = \exp\left(-\frac{(f_x^2(x, y) + f_y^2(x, y))}{\tau^2}\right) \tag{6.10}$$

where f_x and f_y are the usual first derivatives, and they are squared. τ is required to scale the numerator to be in the useful range of the exponential. The VCD algorithm is summarized as follows:

VCD Algorithm

- Compute $c(x, y)$ using Eq. 6.10, or a similar equation, and store it as an image so you can look at it during debugging.
- For every pixel, compute the change in that pixel using Eq. 6.8. Save those values as an image so you can view them. If you have a bug, it will probably show up as zeros in this image.
- Multiply each pixel in the change image by a small number, e.g., $\alpha = 0.01$.
- Add the result of the multiplication to the current value of f to get the next value of f.
- The algorithm may be expressed as a single equation is $f(x, y) \leftarrow f(x, y) + 0.01\nabla^{\mathrm{T}}(c(x, y)\nabla f(x, y))$.

In Figure 6.7 (middle column) results of running VCD on the three test images are illustrated. The algorithm has produced regions with reduced noise and sharp edges. It has modified the clutter substantially. The middle column of Figure 6.8 illustrates the results using random color maps.

One of the problems with VCD is that it is difficult to find a good stopping point without modifying the algorithm. Allowing it to run for too long produces a result such as Figure 6.9.

In this section, we have shown one way to do edge-preserving smoothing by formulating a process, based on a PDE, which does what is desired. There are many variations on edge-sensitive diffusion. Another approach to the same problem is based on minimization of an objective function to obtain equal or better results, and that approach is described in the next section. Before reading it, go back and reread section 3.3 and recall that defining and solving an optimization problem can provide a powerful approach to many applications.

6.5 Edge-Preserving Smoothing Using Optimization

In this section we approach the problem of removing noise from an image by estimating what the image looked like before the noise was added. To do this, an objective function will be derived whose minimum will be found by gradient descent. Several forms for the objective function will be found that will enable the user to specify desired properties of the solution. Implementation of the algorithm presented in this section is enabled by material in Appendix B.

6.5.1 An Objective Function for Noise Removal

This problem will involve several variables:

- $\mathbf{f}$ The image before it was corrupted by noise. It is unknown. Estimation of this image is the problem we must solve. Though it is an image, we construct the lexicographic representation and think of the image as a vector. The elements of $\mathbf{f}$ are then denoted f_i.
- $\mathbf{g}$ The image that we measured. This is corrupted by noise. Its elements are g_i.
- $P(\mathbf{g}|\mathbf{f})$ The conditional probability. Given the (unknown) image $\mathbf{f}$, this is the probability that the image $\mathbf{g}$ will be generated as a result of adding random noise to $\mathbf{f}$.
- $P(\mathbf{f})$ The "prior" or *a priori* probability. This is the probability that $P(\mathbf{f})$ exists. That is, the probability, without having any other information, that the first element of $\mathbf{f}$ has brightness f_0, the second element of $\mathbf{f}$ has brightness f_1, etc. For example, we might know *a priori* that all the images with which we are concerned are almost always locally constant.
- $P(\mathbf{f}|\mathbf{g})$ The posterior probability, or *a posteriori* probability, i.e., the probability that the first element of $\mathbf{f}$ has value $f_0 \ldots$ given the measurement, and we know the measurement. We want to find the particular $\mathbf{f}$ that maximizes this probability. Thus, we call these algorithms *Maximum A Posteriori probability* (MAP) algorithms.
- $P(\mathbf{g})$ The prior probability for $\mathbf{g}$. This depends only on $\mathbf{g}$, and not on $\mathbf{f}$, and $\mathbf{f}$ is what we are trying to find, so this term is of no use. Setting it to one gives a reasonable function to optimize, as shown in Eq. 6.11.

All the terms described above may be related by a single equation, *Bayes rule*:

$$P(\mathbf{f}|\mathbf{g}) = P(\mathbf{g}|\mathbf{f})\frac{P(\mathbf{f})}{P(\mathbf{g})} \; . \tag{6.11}$$

The next step is to find a form for the conditional probability. We start with a model for the distortion process. We assume the image is only distorted by additive noise at every pixel:

$$g_i = f_i + n_i \tag{6.12}$$

so

$$n_i = g_i - f_i \,. \tag{6.13}$$

For a single pixel, assume the noise is Gaussian and

$$P(g_i|f_i) = P(g_i - f_i) = P(n_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(g_i - f_i)^2}{2\sigma^2}\right) \,. \tag{6.14}$$

Assuming that the noise added to one pixel is statistically independent of the noise added to all other pixels allows us to write

$$P(\mathbf{g}|\mathbf{f}) = \prod_i \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(g_i - f_i)^2}{2\sigma^2}\right) \,. \tag{6.15}$$

We are free to use almost anything for the prior, as long as it makes sense, but it is extremely convenient to use some sort of product, since we will be taking logs in the next equation. However there is also theoretical justification, based on the theory of Markov Random Fields (MRF) [6.5, 6.14]. In an MRF, the probability that a pixel has a particular value does depend on the values of its neighbors, which in turn depend on their neighbors, just as you learned for Markov random processes [6.4, 6.11, 6.17]. For now, let's just choose a form for the prior:

$$P(\mathbf{f}) = \prod_i \exp(-\Gamma(f_i)) \,, \tag{6.16}$$

where Γ is some function of the neighborhood of pixel i. A little later, we will choose a form for Γ.

So the function we want to maximize is

$$\prod_i \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(g_i - f_i)^2}{2\sigma^2}\right) \prod_i \exp(-\Gamma(f_i)) \,. \tag{6.17}$$

An expression involving products and exponentials strongly suggests using a logarithm, and doing so produces a new objective, now called H:

$$H = \sum_i \log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \sum_i \frac{(g_i - f_i)^2}{2\sigma^2} - \sum_i \Gamma(f_i) \tag{6.18}$$

The first term in Eq. 6.18 is not dependent on either of the images, so we can ignore it for purposes of optimization. Next, we switch from seeking an image $\mathbf{f}$ that maximizes a probability to seeking an $\mathbf{f}$ that minimizes the objective function by simply changing the signs.

$$H = \sum_i \frac{(g_i - f_i)^2}{2\sigma^2} + \sum_i \Gamma(f_i) \,. \tag{6.19}$$

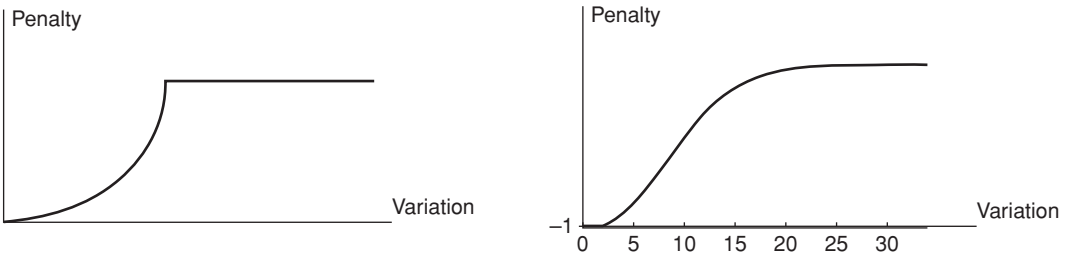


Figure 6.10 The penalty should be stronger for higher noise. Assuming local variations in brightness are due to noise, a larger variation is penalized more. However, local brightness variations can also be due to edges, which should not be penalized (otherwise, they will be blurred). Therefore we want our penalty function to have an upper limit. LEFT: A penalty function that has the right properties, but is not differentiable. RIGHT: A differentiable penalty function that has the same form and is simply the negative of a Gaussian.

The first term H_n, is called the *noise term*, and the second term H_p, the *prior term*.

What does minimizing H mean? We haven't chosen a form for Γ yet, but just looking at the noise term, we notice that it is zero when $\mathbf{f} = \mathbf{g}$, and since both numerator and denominator are positive, it can't get any smaller than zero. Although we started this derivation thinking about probabilities and Gaussian densities, suddenly we have the result that the minimizing $\mathbf{f}$ will resemble the measurement $\mathbf{g}$. This certainly makes intuitive sense.

From the noise term, we already have one requirement to fulfill, that the optimizing $\mathbf{f}$ resemble the measurement $\mathbf{g}$. We use the prior to make the optimizing $\mathbf{f}$ have other properties that we may choose. First, observe that since we are seeking an $\mathbf{f}$ that minimizes H, anything that makes H large is not likely to be chosen. What if we chose something like

$$\Gamma(f_i) = (f_i - f_{i+1})^2 \ ? \tag{6.20}$$

With this Γ, images with a lot of points where the pixel-to-pixel difference is large are not likely to be chosen. We say such points are *penalized*. Could we find an image that minimizes this prior term? Certainly. Any image where the pixel-to-pixel difference is zero would minimize it. Unfortunately, such images are not very interesting.

It's actually even worse! The term $(f_i - f_{i+1})^2$ is *quadratic* in the pixel-to-pixel difference, so it penalizes larger difference even more! But what if it is an edge with a large pixel-to-pixel difference? Using this prior would choose an $\mathbf{f}$ that blurs edges.

We need a term that is small for small differences, grows (perhaps quadratically) with increasing noise, but at some point stops getting larger. Such a function would penalize edges no more than the highest levels of noise. Such a function is shown on the left of Figure 6.10. The problem with that function is that it is discontinuous, and therefore not differentiable, and we will need differentiation later in this section. The function on the right of that same figure however satisfies the same desires, and is continuous. It is simply the negation of a Gaussian.

Using the negative of a Gaussian as the Γ function, the objective function takes on another form:

$$H = \sum_i \frac{(g_i - f_i)^2}{2\sigma^2} - \beta \sum_i \exp\left(-(f_i - f_{i+1})^2\right) \ , \tag{6.21}$$

where $\beta = \frac{b}{\sqrt{2\pi}\tau}$, and b is a weight that can be set to emphasize the importance of the prior vs. the noise terms. Now, we recognize $(f_i - f_{i+1})^2$ as a spatial derivative, and rewrite the objective function, taking into account the fact that we have two spatial derivatives, which are evaluated at pixel i:

$$H = \sum_i \frac{(g_i - f_i)^2}{2\sigma^2} - \beta \sum_i \exp\left(-\left(\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right)\Big|_i\right). \tag{6.22}$$

Finally, we recognize that we need to add a denominator to the argument of the derivative to keep the argument within reasonable bounds:

$$H = \sum_i \frac{(g_i - f_i)^2}{2\sigma^2} - \beta \sum_i \exp\left(-\frac{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}{2\tau^2}\right). \tag{6.23}$$

Now we have an objective function that will produce a smooth image when we find an $\mathbf{f}$ that minimizes it. Minimization will require differentiation. To compute the derivative of H with respect to the elements of $\mathbf{f}$, we make one final modification to Eq. 6.23. We replace the exponential of a sum with the sum of the exponentials. Mathematically, this is a completely different function, yet both prior terms are minimized by a piecewise-constant image, and the second version is significantly easier to work with:

$$H = \sum_i \frac{(g_i - f_i)^2}{2\sigma^2} - \beta \sum_i \left(\exp\left(-\frac{\left(\frac{\partial f}{\partial x}\right)^2}{2\tau^2}\right) + \exp\left(-\frac{\left(\frac{\partial f}{\partial y}\right)^2}{2\tau^2}\right)\right). \tag{6.24}$$

We begin the process of finding the gradient of H by replacing the partial derivatives with convolutions indicating how they will be computed.

$$H = \sum_i \frac{(g_i - f_i)^2}{2\sigma^2} - \beta \sum_i \left\{\exp\left(-\frac{(f \otimes h_x)^2}{2\tau^2}\right) + \exp\left(-\frac{(f \otimes h_y)^2}{2\tau^2}\right)\right\}, \tag{6.25}$$

where h_x and h_y are kernels that estimate derivatives in the x and y directions, respectively.

To minimize the objective function, we use gradient descent, which was covered in section 3.3.1. Gradient descent requires a way to find $\frac{\partial H}{\partial f_i}$ for every pixel, say pixel i. The gradient of Eq. 6.25 for pixel f_i is

$$\begin{aligned}\frac{\partial H}{\partial f_i} = \frac{(f_i - g_i)}{\sigma^2} &+ \beta\left(\frac{1}{\tau^2}\left((f \otimes h_x)\exp\left(-\frac{(f \otimes h_x)^2}{2\tau^2}\right)\right) \otimes h_{xrev}\right)|_{f_i} \\ &+ \beta\left(\frac{1}{\tau^2}\left((f \otimes h_y)\exp\left(-\frac{(f \otimes h_y)^2}{2\tau^2}\right)\right) \otimes h_{yrev}\right)|_{f_i}\end{aligned} \tag{6.26}$$

where the reverse of the kernel h, h_{rev} is a flip of kernel h up to down, left to right. For example,

$$\text{If } h = \begin{array}{|c|c|c|}\hline h_{-1,-1} & h_{-1,0} & h_{-1,1} \\ \hline h_{0,-1} & h_{0,0} & h_{0,1} \\ \hline h_{1,-1} & h_{1,0} & h_{1,1} \\ \hline\end{array}, \text{ then } h_{rev} = \begin{array}{|c|c|c|}\hline h_{1,1} & h_{1,0} & h_{1,-1} \\ \hline h_{0,1} & h_{0,0} & h_{0,-1} \\ \hline h_{-1,1} & h_{-1,0} & h_{-1,-1} \\ \hline\end{array}.$$

This form of the gradient is derived in Appendix B.

The algorithm described here is one of many examples of MAP algorithms, which may be used for image restoration as well as noise removal.

6.5.2 Selecting a Prior Term

Some insight can be gleaned by writing the prior of Eq. 6.21 in a slightly more general way, that is,

$$\Gamma(f_i) = -\frac{1}{\tau}\exp\left(-\frac{(\Lambda(f))_i^2}{2\tau^2}\right) \tag{6.27}$$

where the term $(\Lambda(f))_i$ denotes some function of the (unknown) image f at pixel i. What kind of image minimizes this term? Let's look at what this expression means:

The minus sign[3] in front of the prior term requires that, to minimize the function, we must find the image that causes the exponential to be maximized. To determine what kind of image maximizes an exponential, look at the argument of the exponential. Observe the minus sign and note that both numerator and denominator are squared, and therefore always positive. Thus, the argument of the exponential is always negative. The value of an argument that is always negative and causes the exponential to be maximal is zero. Thus, to maximize the exponential, choose any image f which causes $\Lambda(f)$ to be zero. Maximizing the exponential, in turn, minimizes the objective function.

What do we conclude from this? For any function $\Lambda(f)$, the f that causes $\Lambda(f)$ to be zero is the f that the prior term will seek. This observation gives us a lot of freedom of design. We can choose the function $\Lambda(f)$ to produce the type of solution we seek.

Example: Piecewise-Constant Images

Consider this form for the prior

$$\Lambda(f) = \left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2. \tag{6.28}$$

In order for this term to be zero, both partial derivatives must be zero. The only type of surface that satisfies this condition is one that does not vary in either direction – flat, or flat almost everywhere. To see why the solution is piecewise-constant rather than completely constant, you need to recognize that the total function being minimized is the sum of both the prior and the noise terms. The prior term seeks a constant solution, but the noise term seeks a solution that is faithful to the measurement. The optimal solution to this problem is a solution that is flat in segments, as illustrated in one dimension in red in Figure 6.11.

The function $\Lambda(f)$ is nonzero only in the vicinity of points where f undergoes an abrupt change – an edge. To see more clearly what this produces, consider the extension to continuous functions. If x is continuous, then the summation in Eq. 6.27 becomes an integral. The argument of the integral is nonzero at only a small number of points (referred to as a *set of measure zero*), which is insignificant compared to the rest of the integral.

[3] Most errors are made in implementing this algorithm by dropping minus signs!

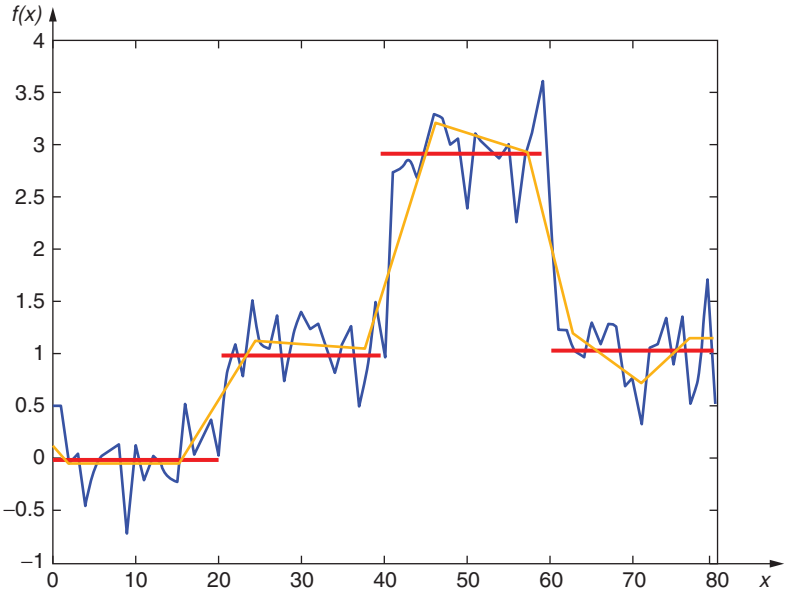


Figure 6.11 Piecewise-constant and piecewise-linear solutions to the fitting of brightness on a single line across a noisy image. The piecewise-constant solution (red) has a derivative equal to zero at almost every point. Nonzero derivatives exist only at the points where steps exist. The piecewise linear solution (orange) has a second derivative equal to zero at almost every point.

Example: Piecewise-Planar Images

Consider

$$\Lambda^2(f) = \left(\frac{\partial^2 f}{\partial x^2}\right)^2 + \left(\frac{\partial^2 f}{\partial y^2}\right)^2 + 2\left(\frac{\partial^2 f}{\partial x \partial y}\right)^2 . \tag{6.29}$$

What does this do? What kind of function has a zero for all its second derivatives? The answer is a plane. Thus, using this form[4] for $\Lambda(f)$ will produce an image that is locally planar, but still maintains fidelity to the data – a piecewise-planar image. Another alternative operator that is also based on the second derivative is the Laplacian, $\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$. We saw both of these back in Chapter 5.

You might ask your instructor, "Breaking a brightness image into piecewise-linear segments is the same as assuming the actual surfaces are planar, right?" The answer would probably be "Yes, except for variations in lighting, reflectance, and albedo." Ignoring that, you charge on, saying "But real surfaces aren't all planar." The answer to your concern is two-fold: First is the trivial and useless observation that all surfaces *are* planar – you just have to examine a sufficiently small region. But second, you should realize that you will never run the algorithm long enough to converge to a perfect piecewise-linear solution, but only far enough to get sufficient smoothing.

A one-dimensional example of a piecewise linear estimate is shown in orange in Figure 6.11.

[4] This is the *quadratic variation*. The Laplacian also involves second derivatives. Ask yourself how it differs from the quadratic variation.

In conclusion, you can choose any function for the argument of the exponential that you wish, as long as the image you want is produced by setting that argument to zero. Some more general properties of prior models have been presented by Li [6.18]: As a function of the local image gradient Λ, a prior should

1. be continuous in its first derivative,
2. be even ($h(\Lambda) = h(-\Lambda)$),
3. be positive $h(\Lambda) > 0$,
4. have a negative derivative for positive argument $h(\Lambda) < 0$ for $\Lambda > 0$, and
5. go to a constant in the limit $\lim_{\Lambda \to \infty} |h(\Lambda)| = C$.

Interestingly, Yi and Chelberg [6.28] observe that second-order priors require a great deal more computation than first-order priors, and that first-order priors can be made approximately invariant. However, in our own experiments, we have found that the increased flexibility provided by second order priors outweighs the computational penalty imposed by their use.

6.5.3 MAP Algorithm Implementation and Mean Field Annealing (MFA)

The implementation details of the MAP algorithm are listed in Figure 6.12 on page 110:

Implementing the MAP algorithm based on gradient descent often suffers from local minima, as discussed in section 3.3.2. This can be improved by adding *annealing*. Mean Field Annealing (MFA) is a technique for finding a good minimum of complex functions that typically have many minima. The mean field approximation of statistical mechanics allows a continuous representation of the energy states of a collection of particles. In the same sense, MFA approximates the stochastic algorithm called "simulated annealing" (SA). SA has been shown to converge in probability to the global minimum, even for non-convex problems [6.11]. Because SA also takes an unacceptably long time to converge, a number of techniques have been derived to accomplish speed-ups [6.16]; MFA is one of those techniques.

Adding annealing to the previous algorithm gives us one more capability that distinguishes it from other MAP methods, the ability to avoid most local minima. The "magic" is all in the use of the parameter τ in Eq. 6.27. Figure 6.12 is the entire modified algorithm, after the gradient of the objective function, $\nabla_f(H(f))$, has been found analytically.

The details of why this process of annealing avoids local minima are described elsewhere [6.6, 6.7, 6.8], but this process results from the analogy to simulated annealing [6.10] (see section 3.3.3).

Let's see what the words in quotes in the MFA algorithm on page 110 mean.

Consider what happens if τ_{init} is large. A large τ will cause the argument of the exponential to be close to zero, and the value of the exponential itself to be approximately one. But that value is itself divided by τ, so that when τ is large, the value of the prior is on the order of $1/\tau$; and if τ is a big number, then the prior is insignificant relative to the noise term. We can ensure that τ is initially "large" by choosing τ_{init} to be larger than, say, twice the average value of the numerator,

$$\tau_{init} = 2 < |f \otimes r| >, \tag{6.30}$$

1. Gradient descent is used to modify the current estimate of f, using the derivatives of the H_n and H_p. The derivative of the noise term is trivial: on each iteration, simply change pixel i by $\Delta H_{n_i} = \alpha \frac{(f_i - g_i)}{\sigma^2}$, for some small constant α.
2. Three kernel operators may be used to estimate the three partial second derivatives in the quadratic variation
$$\Delta_{xx} = \frac{1}{6}\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \Delta_{yy} = \frac{1}{6}\begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} \Delta_{xy} = 2\begin{bmatrix} -0.25 & 0 & 0.25 \\ 0 & 0 & 0 \\ 0.25 & 0 & -0.25 \end{bmatrix}.$$
3. Compute three images, where the i^{th} pixels of those images are $r_{ixx} = (\Delta_{xx} \otimes f)_i$, $r_{iyy} = (\Delta_{yy} \otimes f)_i$, and $r_{ixy} = (\Delta_{xy} \otimes f)_i$.
4. Create the image s_{xx}, whose elements are $\frac{r_{ixx}}{\tau} \exp\left(-\frac{r_{ixx}^2}{2\tau^2}\right)$, and similarly create s_{yy} and s_{xy}.
5. For purposes of gradient descent, the change in pixel i from the prior term is then $\Delta H_{p_i} = \beta((\Delta_{xx} \otimes s_{xx})_i + (\Delta_{yy} \otimes s_{yy})_i + (\Delta_{xy} \otimes s_{xy})_i)$.
6. Following the gradient descent rule, change each element of f using $f_i \leftarrow f_i - \eta \Delta H_i$, where $\Delta H_i = \Delta H_{n_i} + \Delta H_{p_i}$, for some small scalar constant η.
7. The *learning coefficient* η can be shown to be $\eta = \frac{\gamma \sigma \sqrt{t}}{RMS(\Delta H_i)}$, where γ is a small dimensionless number, like 0.04; $RMS(\Delta H)$ is the root mean square norm of the gradient ΔH; and σ can be determined as the standard deviation of the noise in the image (note that this is NOT necessarily a good estimate in synthetic images). In this form, η changes every iteration. However, η could also simply be chosen to be some small number like .01.
8. The coefficient β is on the order of σ, and choosing $\beta = \sigma$ is usually adequate.

Figure 6.12 The MAP algorithm using a piecewise-linear prior.

where r represents a derivative kernel applied on image f. Having decided to choose an initial value of τ to make the prior zero, the resulting initial objective function is just the noise term, and it is minimized by choosing an initial value of f to be g.

MFA is based on the mathematics of simulated annealing. One can show that in simulated annealing, a global minimum can be achieved by following a logarithmic annealing schedule like

$$\tau(k) = \frac{1}{\ln k} \tag{6.31}$$

MAP with annealing

1. Begin with "large" τ.
2. Do gradient descent to get increasingly closer to the nearest local minimum.
3. As the descent iterations proceed, "slowly" reduce τ.
4. When τ reaches an application-dependent small value, stop.

Figure 6.13 Outline of the use of MAP with Annealing.

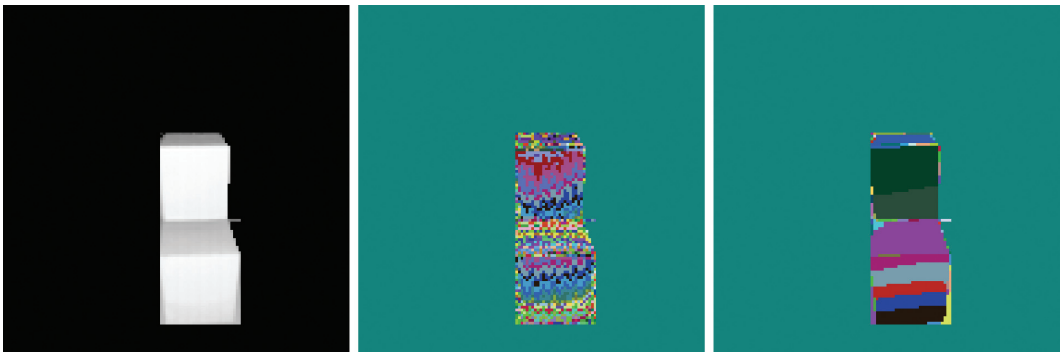


Figure 6.14 LEFT: The cropped image of two boxes, scanned with a laser scanner. This image shows the strength of the return signal from the laser. MIDDLE: The noise in the image illustrated using the random colormap. Almost every pixel varies from its neighbor by at least one brightness level. RIGHT: Noise removal by the MFA algorithm using a piecewise-constant prior, where variations of the brightness due to noise are eliminated.

where k is the iteration number. This schedule decreases τ extremely slowly – so slowly as to be impractical. Instead, one could choose a schedule like

$$\tau(k) = 0.99\tau(k-1) \tag{6.32}$$

which has been shown to work satisfactorily in many applications, and reduces τ much faster than the logarithmic schedule. However, the guarantee of converging to the global minimum is lost. We must be satisfied with a good minimum, which is almost always sufficient.

Annealing over a couple of orders of magnitude of τ produces the reductions of noise demonstrated in the rightmost columns of Figures 6.7 and 6.8.

Finally, the range image shown in Figure 4.1 is presented in Figure 6.14, zoomed and smoothed with MFA.

6.5.4 Ill-Conditioned Inverse Problems and Regularization

A problem $\mathbf{g} = D(\mathbf{f})$ is said [6.13] to be well-posed if

- For each $\mathbf{g}$, a solution, $\mathbf{f}$, exists
- The solution is unique
- The solution $\mathbf{f}$ continuously depends on the data $\mathbf{g}$.

If these three conditions do not all hold, the problem is said to be "ill-conditioned" or "ill-posed." Conditioning of a mathematical problem is measured by the sensitivity of output to changes in input. For a well-conditioned problem, a small change of input does not affect the output much, while for an ill-conditioned problem, a small change of input can change the output a great deal.

An example of ill-conditioning [6.15] is as follows: Consider the two-variable linear system described by a matrix A, an unknown image $\mathbf{f}$, and a measurement $\mathbf{g}$, where

$$\mathbf{g} = A\mathbf{f} \quad A = \begin{bmatrix} 1 & 1 \\ 1 & 1.01 \end{bmatrix} \quad \mathbf{f} = [f_1 \; f_2]^{\mathrm{T}} \quad \mathbf{g} = [1 \;\; 1]^{\mathrm{T}} \; .$$

This inverse problem has solution $f_1 = 1, f_2 = 0$. Now, suppose the measurement, $\mathbf{g}$, is corrupted by noise, producing $\mathbf{g} = [1 \;\; 1.01]^{\mathrm{T}}$. Then, the solution to the inverse problem is $f_1 = 0, f_2 = 1$. A trivially small change in the measured data caused a dramatic change in the solution.

A rigorous exploration into the nature of ill-conditioned problems is outside the scope of this book, but the student should be aware that many inverse problems are ill-conditioned. In particular, determining the input to a system with noise by using only the output of the system is frequently ill-conditioned.

There are many ways to approach these ill-conditioned restoration problems. They all share a common structure: regularization. Generally speaking, any regularization method tries to find a related well-posed problem whose solution approximates the solution to the original ill-conditioned problem [6.20]. One of the more common regularization methods is to add an additional term to the objective function that limits the nature of the solution. In some problems, this might be a constraint, and in some terms a penalty. As we progress through the book, the difference between a penalty and a constraint will become clear. In this chapter, we see a penalty: the unknown image should be smooth, and resemble the measurement; otherwise, a penalty will be imposed. More recently developed penalty terms for denoising purpose include the total variation [6.25] and the nonlocal means [6.9]. In later chapters, we will make use of a constraint, e.g., a vector MUST be a unit vector.

6.6 Equivalent Algorithms

In this section, you will learn the surprising result that using diffusion for noise removal and the MFA algorithm we have discussed are really the same algorithm. Use of diffusion results from thinking about the problem as a process, in this case diffusion. MFA results from thinking about the problem as finding the minimum of some objective function.

For purposes of this derivation, only derivatives in the x-direction are considered and only the prior term is considered. This will be generalized to two dimensions after we complete the derivation. Rewrite the prior term as

$$H_p(f) = \frac{-b}{\sqrt{2\pi}\,\tau} \sum_i \exp\left(-\frac{(f \otimes r)_i^2}{2\tau^2}\right) \; . \tag{6.33}$$

The notation $(f \otimes r)_i$ means the result of applying a kernel r to the image f at point i. The kernel may be chosen to emphasize problem-dependent image characteristics. This general form was used earlier to remove noise from piecewise-constant and piecewise-linear images.

The derivative is required to perform gradient descent with respect to the unknown image f so we calculate (from Eq. B.8):

$$\frac{\partial H_p}{\partial f_i} = \frac{b}{\sqrt{2\pi}\tau}\left(\left(\frac{(f \otimes r)}{\tau^2}\exp\left(-\frac{(f \otimes r)^2}{2\tau^2}\right)\right) \otimes r_{rev}\right)_i \tag{6.34}$$

where r_{rev} denotes the mirror image of the kernel r. In order to follow the derivation most easily, think of r as the kernel that estimates the first derivative. Then $(f \otimes r)_i$ can be written $(\nabla f)_i$. Below, the subscript i will be dropped just to keep the number of parentheses manageable.

We seek to make the magnitude of the gradient of the image small almost everywhere. To get insight into what is going on, replace some of the convolution notation in Eq. 6.34 by gradients:

$$\frac{\partial H_p}{\partial f_i} = \kappa((\nabla f)\exp(-(\nabla f)^2)) \otimes r_{rev} \tag{6.35}$$

and then

$$\frac{\partial H_p}{\partial f_i} = -\kappa(\nabla((\nabla f)\exp(-(\nabla f)^2))) \; . \tag{6.36}$$

In the equation above, the constants are lumped together into κ; and the annealing control parameter, τ, is set to 1 for clarity. Note that for centered first derivative kernels, $f \otimes r = -(f \otimes r_{rev})$.

Finally, consider the use of $\frac{\partial H_p}{\partial f_i}$ in a gradient descent algorithm. In the simplest implementations of gradient descent, f is updated by

$$f_i^{k+1} = f_i^k - \eta\frac{\partial H_p}{\partial f_i} \tag{6.37}$$

where f^k denotes the value of f at iteration k, and η is some small constant. Rewriting Eq. 6.37,

$$-\frac{\partial H_p}{\partial f_i} = \frac{f_i^{k+1} - f_i^k}{\eta} \; . \tag{6.38}$$

Note that the left-hand side of Eq. 6.38 represents a change in f between iterations k and $k+1$, and in fact bears a strong resemblance to the form of the derivative of f. We make this similarity explicit by defining that iteration k is calculated at time t and iteration $k+1$ calculated at time $t + \Delta t$. Since t is an artificially introduced parameter, without physically meaningful units, we may scale it by any convenient proportionality constant, and we obtain

$$-\frac{\partial H_p}{\partial f_i} = \frac{f_i(t+\Delta t) - f_i(t)}{\Delta t} \approx \frac{\partial f_i}{\partial t} \; , \tag{6.39}$$

where we have allowed the constant α to be renamed Δt so that the expression looks like a derivative with respect to time. Substituting this (re)definition into Eq. 6.35, we have our final result: By simply changing notation and making explicit the time between iterations, the derivative of the MAP prior term may be rewritten as

$$\frac{\partial f_i}{\partial t} = \kappa(\nabla((\nabla f)\exp(-(\nabla f)^2))) \; . \tag{6.40}$$

You saw the diffusion equation

$$\frac{\partial f_i}{\partial t} = \kappa(\nabla(c\nabla f)) \tag{6.41}$$

previously in section 6.4. Observe that this is identical to Eq. 6.40 when the conductivity, c, is replaced by the exponential.

If image brightness is thought of as heat and allowed to diffuse over time (iterations), the result smoothes out noise, and if c is properly computed, edges can be preserved as well. We already saw Variable Conductance Diffusion (VCD) [6.12, 6.21, 6.23] in section 6.4.4, which is this exact algorithm.

By Eq. 6.40, we have shown the equivalence of MAP and VCD, provided that MAP is performed without using the noise term. This equivalence provides for the union of two schools of thought concerning image analysis: The *optimization* school considers the properties that an image ought to have. It then sets up an optimization problem whose solution is the desired image. One might also call this the restoration school. The *process* school is more concerned with determining which spatial analysis methods to apply. Adaptive filtering, diffusion, template matching, etc. are more concerned with the process itself than with what that process does to some hypothetical "energy function" of the image. The results shown in this section demonstrate that these two schools are not just philosophically equivalent. At least for this particular form of edge-preserving smoothing, they are precisely equivalent.

The student may ask, "At least from the figures, it appears that VCD does not work as well as MAP, but how can that be, if they are equivalent?" The answer is that the above equivalence considers only the prior term of the MAP objective function. Addition of the noise term adds the desire to not only be smooth with sharp edges, but also to resemble the original. Nordström [6.21] also observed a similarity between diffusion techniques and regularization (optimization) methods. He notes that "the anisotropic diffusion method does not seek an optimal solution of any kind." Nordström then proceeds to "unify from the original outlook quite different methods of regularization and anisotropic diffusion." He proceeds quite elegantly and precisely to derive a cost function whose behavior is an anisotropic diffusion, in a similar manner to the derivation presented here. Nordström also argues for the necessity of adding a "stabilizing cost" to "restrict the space of possible estimated image functions." At this point the reader should not be surprised to learn that the form of the stabilized cost is

$$\sum_i (f_i - g_i)^2 \ , \tag{6.42}$$

which we showed in Eq. 6.19 to be a measure of the effect of Gaussian noise in a blur-free imaging system. Thus we see that Biased Anisotropic Diffusion (BAD) [6.21] may be thought of as a maximum *a posteriori* restoration of an image. This observation now permits researchers in VCD/BAD to consider use of different stabilizing costs, if they have additional information concerning the noise generation process.

6.7 Conclusion

In this chapter we have, among other things, presented three approaches to the same problem: edge-preserving smoothing. Our purpose was to use this as an opportunity to teach three different philosophies of solving problems in Computer Vision: (1) develop an ad hoc "good idea" into an effective algorithm, (2) pose the problem as the solution to a differential equation, or (3) pose the problem as a minimization. With proper choice of parameters, the three methods produce similar results. MAP was shown to perform better, but that is because MAP not only smooths the image; it also maintains fidelity to the original image. It is often easier to pose Computer Vision problems as solutions to optimization problems.

The strategy of bilateral filtering first appeared in the literature 1995 in a paper [6.1] by Aurich and Weule, but the term "bilateral filtering" did not appear until 1998 [6.26]. It has been used in applications such as noise removal [6.2, 6.3, 6.19], image motion analysis [6.27], and many others. The authors of this book used it in converting from the image representation in digital cameras to conventional color, a process called demosaicking [6.24].

Optimization methods are so pervasive in this chapter that the chapter title could almost be "image optimization." We set up an objective function, a function of the measured image and the (unknown) true image and find the (unknown) true image that minimizes the objective function. Two terms were introduced: a noise term, which depends on the measurement; and a prior term, which depends only on the true image. The "true" image is then determined by finding the image that minimizes the objective function. A variety of minimization techniques could be used. In this chapter, we use gradient descent with annealing, but other, more sophisticated and faster techniques, such as conjugate gradient, could be used.

6.8 Assignments

Assignment 6.1: Implement the VCD algorithm in section 6.4.4 on an image that your instructor selects. Experiment with various run times and parameter settings.

Assignment 6.2: In Eq. 6.29, the quadratic variation is presented as a prior term. A very similar prior would be the Laplacian. What is the difference? That is, are there image features that would minimize the Laplacian and not minimize the quadratic variation? Vice-versa? Note that we said "the Laplacian" and not "the squared Laplacian." Comment on this.

Assignment 6.3: Which of the following expressions represents the Laplacian?

$$\text{(a) } \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \qquad \text{(c) } \left(\frac{\partial^2 f}{\partial x^2 f}\right)^2 + \left(\frac{\partial^2 f}{\partial y^2}\right)^2 \qquad \text{(e) } \sqrt{\left(\frac{\partial^2 f}{\partial x^2}\right)^2 + \left(\frac{\partial^2 f}{\partial y^2}\right)^2}$$

$$\text{(b) } \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \qquad \text{(d) } \left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial x}\right)^2 \qquad \text{(f) } \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Assignment 6.4: One form of the diffusion equation is written as $df/dt = h_x \otimes (c(h_x \otimes f)) + h_y \otimes (c(h_y \otimes f))$ where h_x and h_y estimate the first derivatives in the x and y directions, respectively. This suggests that four kernels must be applied to compute this result. Simple algebra, however, suggests that this could be rewritten as $df/dt = c(h_{xx} \otimes f + h_{yy} \otimes f)$, which requires application of only two kernels. Is this simplification of the algorithm correct? If not, explain why not, and under what conditions it would be true.

Assignment 6.5: Consider the following image objective function

$$H(f) = \sum_i \left(\frac{f_i - g_i}{\sigma^2} \right)^2 - \sum_i \exp\left(-\frac{(h \otimes f)_i^2}{\tau^2} \right) = H_n(f, g) + H_p(f) \, .$$

The pixels of f are lexicographically indexed by i, and the kernel h is $\begin{array}{|c|c|c|}\hline -1 & 2 & -1 \\ \hline -2 & 4 & -2 \\ \hline -1 & 2 & -1 \\ \hline \end{array}$. Let $G_p(f_k)$ denote the partial derivative of H_p with respect to pixel k, i.e., $G_p(f_k) = \frac{\partial}{\partial f_k} H_p(f)$. Write an expression for $G_p(f_k)$. Use kernel notation.

Assignment 6.6: Continuing Assignment 6.5, consider *only* the prior term. Write an equation that describes the change in image brightness at pixel k as one iteration of a simple gradient descent algorithm.

Assignment 6.7: Continuing Assignment 6.6, expand this differential equation (assume the brightness varies only in the x direction) by substituting the form for $G_p(f_k)$ that you derived. Is this a type of diffusion equation? Discuss. (Hint: replace the application of kernels with appropriate derivatives).

Assignment 6.8: In a diffusion problem, you are asked to diffuse a VECTOR quantity as in Eq. 6.8, instead of the brightness that you did in your project. Replace the terms in the diffusion equation with the appropriate vector quantities, and write the new differential equation. (Hint, you may find the algebra easier if you denote the vector as $[a \quad b]^{\mathrm{T}}$).

Assignment 6.9: The time that the diffusion runs is somehow related to blur. This is why some people refer to diffusions of this type as "scale-space." Discuss this use of terminology.

Assignment 6.10: On page 109, a list of properties is given that a prior function should have. Determine if the inverted Gaussian has these properties.

Assignment 6.11: In Eq. B.7 of Appendix B, there is a minus sign in front of the sum, but in the next equation, that minus sign is gone. Where did it go?

Assignment 6.12: I had an image that needed to be blurred, so I ran flDoG on it to convolve with a Gaussian of $\sigma = 2$, but that wasn't enough blur. So I then ran the same function, with $\sigma = 2$ again, and the results were exactly what I wanted. Could I have gotten the same result by just running the program once? If so, what value of σ should I have used?

Assignment 6.13: Early in this chapter, an equation is given that relates two Gaussian blurs to a single blur. Suppose you have already blurred an image using convolution with a Gaussian with $\sigma = 0.5$. It turns out you actually needed to blur the original image with a

Gaussian with a σ of 2.0. You don't have access to the original image, so how much more blurring should you do to the one image you already have?

Assignment 6.14: In Appendix B, Eq. B.3 illustrates the partial derivative of an expression involving a kernel, by expanding the kernel into a sum. Use this approach to prove that Eq. B.8 can be derived from Eq. B.7. Do your proof using a one-dimensional problem, and use a kernel that is 3×1. (Denote the elements of the kernel as h_{-1}, h_0, and h_1).

Bibliography

[6.1] Non-linear Gaussian filters performing edge preserving diffusion. In *Proceedings of the DAGM Symposium*, 1995.

[6.2] M. Aleksic, M. Smirnov, and S. Goma. Novel bilateral filter approach: Image noise reduction with sharpening. In *Proceedings of the Digital Photography II Conference*, volume 6069. SPIE, 2006.

[6.3] E. Bennett and L. McMillan. Video enhancement using per-pixel virtual exposures. In *Proceedings of the ACM SIGGRAPH conference*, 2005.

[6.4] J. Besag. Spatial interaction and the statistical analysis of lattice systems. *J Royal Stat. Soc.*, 36, 1974.

[6.5] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society*, 48(3), 1986.

[6.6] G. Bilbro and W. Snyder. Range image restoration using mean field annealing. In *Advances in Neural Network Information Processing Systems*. Morgan-Kaufmann, 1989.

[6.7] G. Bilbro and W. Snyder. Mean field annealing, an application to image noise removal. *Journal of Neural Network Computing*, 1990.

[6.8] G. Bilbro and W. Snyder. Optimization of functions with many minima. *IEEE Transactions on SMC*, 21(4), July/August 1991.

[6.9] A. Buades, B. Coll, and J. M. Morel. A non-local algorithm for image denoising. In *CVPR*, 2005.

[6.10] D. Geiger and F. Girosi. Parallel and deterministic algorithms for mrfs: Surface reconstruction and integration. *AI Memo*, (1114), 1989.

[6.11] D. Geman and S. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. and Machine Intel.*, 6(6), November 1984.

[6.12] S. Grossberg. Neural dynamics of brightness perception: Features, boundaries, diffusion, and resonance. *Perception and Psychophysics*, 36(5), 1984.

[6.13] J. Hadamard. *Lectures on the Cauchy Problem in Linear Partial Differential Equations*. Yale University Press, 1923.

[6.14] J. Hammersley and P. Clifford. Markov field on finite graphs and lattices. Unpublished manuscript.

[6.15] E. Hensel. *Inverse Theory and Applications for Engineers*. Prentice-Hall, 1991.

[6.16] S. Kapoor, P. Mundkur, and U. Desai. Depth and image recovery using a MRF model. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(11), 1994.

[6.17] R. Kashyap and R. Chellappa. Estimation and choice of neighbors in spatial-interaction model of images. *IEEE Trans. Information Theory*, IT-29, January 1983.

[6.18] S. Li. On discontinuity-adaptive smoothness priors in computer vision. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(6), 1995.

[6.19] C. Liu, W. Freeman, R. Szeliski, and S. Kang. Noise estimation from a single image. In *Proceedings of the Conference on IEEE Computer Vision and Pattern Recognition*, 2006.

[6.20] M. Nashed. Aspects of generalized inverses in analysis and regularization. *Generalized Inverses and Applications*, 1976.

[6.21] N. Nordström. Biased anisotropic diffusion-a unified regularization and diffusion approach to edge detection. *Image and Vision Computing*, 8(4), 1990.

[6.22] S. Paris, P. Kornbrobst, J. Tumblin, and F. Durand. A gentle introduction to bilateral filtering and its applications. In *ACM SIGGRAPH 2008*, 2008.

[6.23] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 12, 1990.

[6.24] R. Ramanath and W. Snyder. Adaptive demosaicking. *Journal of Electronic Imaging*, 12(4), 2003.

[6.25] L. Rudin, S. J. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D.*, 60, 1992.

[6.26] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *International Conference on Computer Vision*, 1998.

[6.27] J. Xiao, H. Cheng, H. Sawhney, C. Rao, and M. Isnardi. Bilateral filtering-based optical flow estimation with occlusion detection. In *Proceedings of the European Conference on Computer Vision*, 2006.

[6.28] J. Yi and D. Chelberg. Discontinuity-preserving and viewpoint invariant reconstruction of visible surfaces using a first-order regularization. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(6), 1995.

7 Mathematical Morphology

I think the universe is pure geometry – basically, a beautiful shape twisting around and dancing over space-time. – Antony Garrett Lisi

7.1 Introduction

In this chapter, a study of shape begins. Throughout the remainder of this book we will continue to look at shape in many different ways. Here, the topic is introduced at the pixel level, providing some simple, local operations that modify shapes, changing connectivity, and providing nonlinear[1] ways to treat noise.

- (Section 7.2) Binary morphology changes the shape of binary objects. We define the two basic morphological operators, dilation and erosion, on the basis of which two other operators, opening and closing, can be derived. We also discuss some interesting properties of these morphological operators and their applications in noise removal and edge linking.
- (Section 7.3) We extend the discussion to grayscale morphology and define the dilation and erosion operations when the objects are not described in binary values. We progress from grayscale morphology using flat structuring elements to the use of grayscale structuring elements.
- (Section 7.4) As one of the most important applications of morphological operations, we discuss the distance transform (DT) and different ways to compute the DT.

Morphological operations may be considered as either the result of set operations, or the result of composition of functions. Both these approaches are described in this chapter, in order to provide the student with the insight to see similarities and suitability. We will find that set-theoretic ways of thinking about morphology are most appropriate for binary images, and functional notation is more convenient for grayscale images. In section 7.5, an application example of combining dilation, erosion, and distance transform for the purpose of edge linking is provided to illustrate the power of these operations.

[1] That is, analogous to set intersection and union.

7.2 Binary Morphology

In this section, we first introduce two basic morphological operators, dilation and erosion. We then define opening and closing, two operators that are developed based on dilation and erosion. We also discuss interesting properties of these operators and their applications in solving computer vision problems.

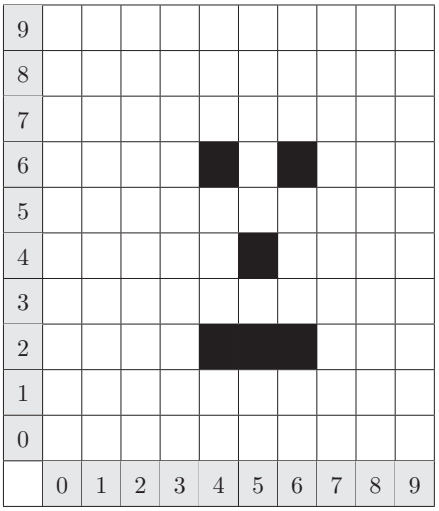
7.2.1 Dilation

First, we present the intuitive definition: the dilation of an image is that same image with all the foreground regions made just a little bit bigger.

Initially, we consider only binary images with foreground pixels having brightness 1 and background 0. With this definition, the foreground may be considered as a set of ordered pairs, consisting of the coordinates of each foreground pixel in the image. For example

$$A = \{(4, 6), (6, 6), (5, 4), (4, 2), (5, 2), (6, 2)\} . \tag{7.1}$$

is a set representation of the image below (where coordinates are shaded).



Let us define a second image with only two pixels,

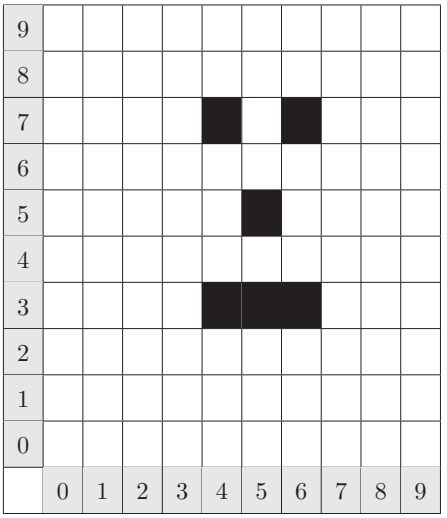
$$B = \{(0, 0), (0, 1)\} \tag{7.2}$$

Consider one pixel in B and call that element $b \in B$. Create a new set by adding the ordered pair b to *every* ordered pair in A.

For example, for the image A, adding the pair (0,1) results in the set

$$A_{(0,1)} = \{(4, 7), (6, 7), (5, 5), (4, 3), (5, 3), (6, 3)\}.$$

The corresponding image is

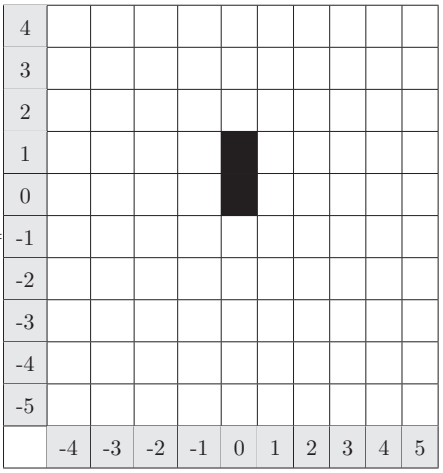


By now, you have probably observed that $A_{(0,1)}$ is nothing more than a translation of A. With this concept firmly understood, think about what would happen if you constructed a *set* of translations of A, one for each pair in B. We denote that set as $\{A_b, b \in B\}$, that is, b is one of the ordered pairs in B.

Formally, the *dilation* of A by B is defined as $A \oplus B = \{a + b \mid (a \in A, b \in B)\}$, which is the same as the union of all those translations of A,

$$A \oplus B = \bigcup_{b \in B} A_b \, . \tag{7.3}$$

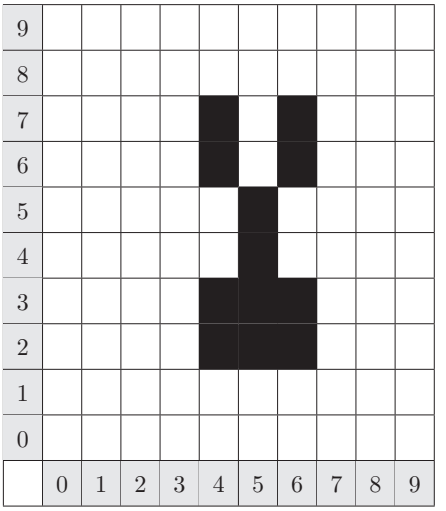
Using these two sets, A and B, and thinking of $B = \{(0, 0), (0, 1)\}$ as an image,

$B =$  .

The resulting union produces

$$A \oplus B = A_{(0,0)} \cup A_{(0,1)} = \{(4, 6), (6, 6), (5, 4), (4, 2), (5, 2), (6, 2), (4, 7), (6, 7), (5, 5), (4, 3), (5, 3), (6, 3)\}$$

which, represented as an image, is

$A \oplus B =$ 

.

The number of elements in set A is denoted by $\#A$, (the *cardinality* of the set). In this example, $\#A$=6, $\#B$=2, and $\#(A \oplus B)$=12. This happens to be true only coincidentally, because there was no overlap between $A_{(0,0)}$ and $A_{(0,1)}$, or said another way: $A_{(0,0)} \cap A_{(0,1)} = \emptyset$. For a more general problem, this will not be the case. In general,

$$\#(A \oplus B) \leq \#A \cdot \#B \ . \tag{7.4}$$

You will observe that usually (for all practical purposes) one of the images is "small" compared to the other. That is, in the example above

$$\#A \gg \#B \ . \tag{7.5}$$

When this is the case, we refer to the smaller image B, as the *structuring element* (s.e.). Observe that all images (whether data images or structuring elements) are based on coordinate systems, and coordinates can go positive or negative. The special pixel located at (0,0) is referred to as the *origin*.

We will frequently use structuring elements, and will often make use of the origin. All the points in an s.e. are relative to some origin, but it is important to note that *the origin is not necessarily in the s.e*. For example, consider the s.e. $\{(1,0),(0,1),(-1,0),(0,-1)\}$ that has four points, none of which are (0,0).

Figure 7.1 illustrates a binary image, its dilation, and its erosion using the following structuring element:

0	1	0
1	1	1
0	1	0

.

In the following material, we will no longer distinguish between the set notation and the image represented by that set. Figure 7.2 shows a code clip for implementing dilation.

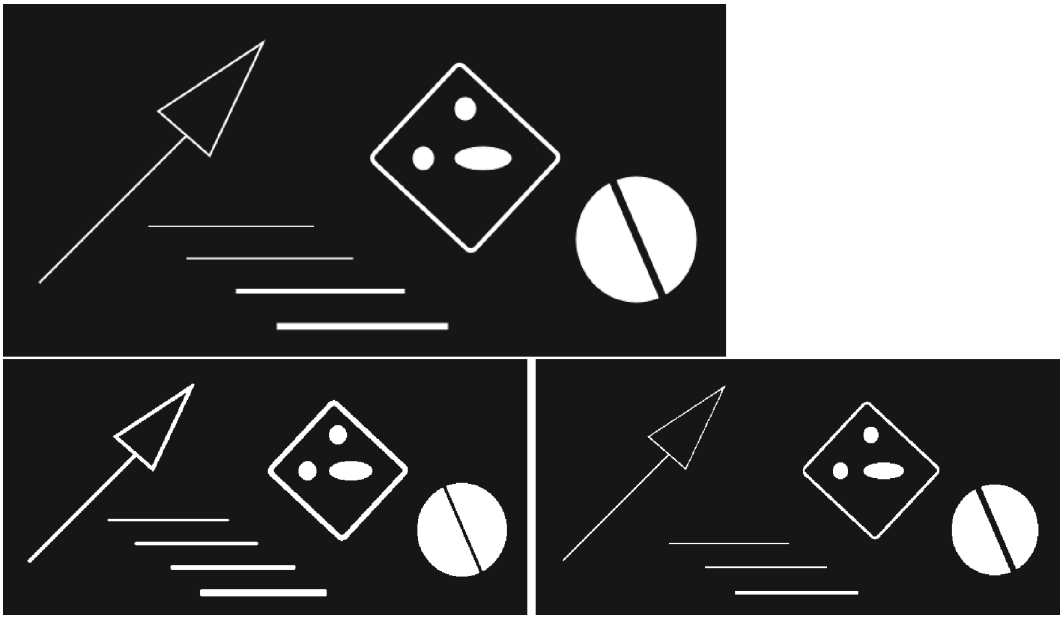


Figure 7.1 A binary figure (above) and its dilation and erosion by a 3×3 structuring element.

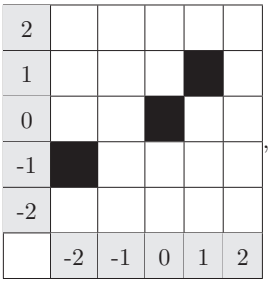
```
// Code Clip: Binary Dilation

#define GOOD(r,c) ((r >= 0) && (r < nr) && (c >= 0) && (c < nc))
int bdila(float **inp, float **sep, float **oup, int nr, int nc,
            int nser, int nsec, int orgr, int orgc, float th)
{
    int r, c, i, j, I, J;

    for (r = orgr; r < nr; r++)
        for (c = orgc; c < nc; c++)
        {
            if (inp[r][c] > th)
            {
                for (i = 0; i < nser; i++) // i is same direction as r
                    for (j = 0; j < nsec; j++) // j is same direction as c
                        if (sep[i][j] > th)
                        {
                            I = i-orgr;
                            J = j-orgc;
                            if (GOOD(r,c))
                                oup[r-I][c-J] = ONE;
                        }
            }
            return 0;
        }
}
```

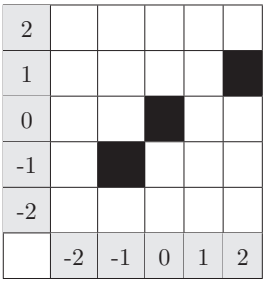
Figure 7.2 A function that computes binary dilation of input image `inp` by s.e. `sep` producing output `oup`. `r` and `c` denote rows and columns in the image, whereas `i` and `j` denote rows and columns in the structuring element. `orgr` and `orgc` are the coordinates of the origin of the structuring element. `th` is the threshold that defines foreground as a point with brightness greater than `th`.

To go further, we need to define some notation: If x is an ordered pair:

1. The *translation*[2] of a set A by x is A_x,
2. The reflection of A is denoted as $\tilde{A}$ and $\tilde{A} = \{(-x, -y) \mid (x, y) \in A\}$.

An example of reflection is

$A =$ 

, $\tilde{A} =$ 

.

Here, $A = \{(0, 0), (-2, -1), (1, 1)\}$ and the reflection of A is $\tilde{A} = \{(0, 0), (2, 1), (-1, -1)\}$.

7.2.2 Erosion

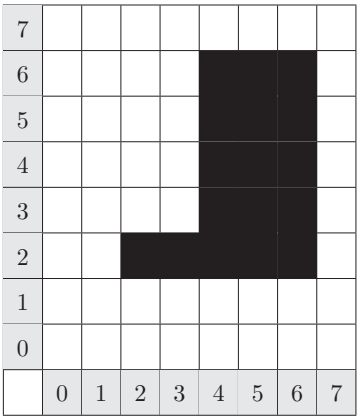
The erosion operator is defined to be a sort of inverse of dilation. It is not a formal inverse because dilation followed by erosion does not necessarily produce the original image. In fact, this is one of the most powerful aspects of morphological operations.

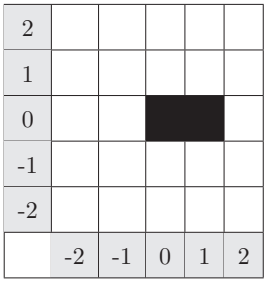
$$A \ominus B = \{a \mid (a + b) \in A \text{ for every } (a \in A, b \in B)\} \tag{7.6}$$

which can be written in terms of translations by

$$A \ominus B = \bigcap_{b \in \tilde{B}} A_b \, . \tag{7.7}$$

Notice two things: the second set, B, is reflected, and the intersection symbol is used. Let's do an example:

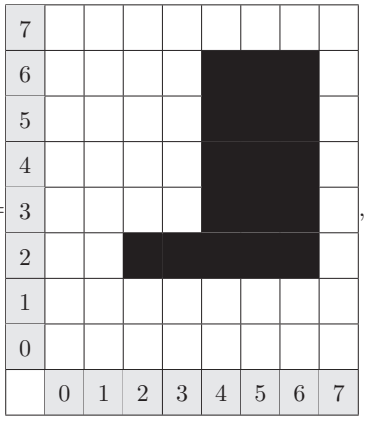
$A =$ 

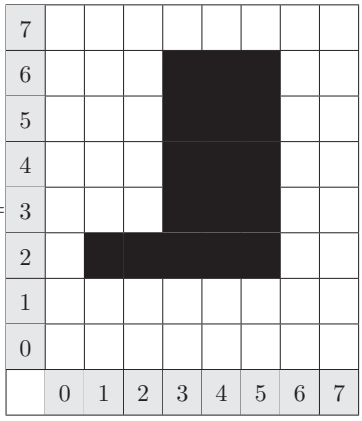
and $B =$ 

.

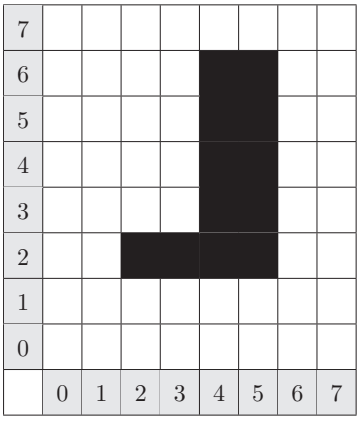
[2] We do not have to be limited to a 2-space. As long as x and A are drawn from the same space, it works. More generally, if A and B are sets in a space ε, and $x \in \varepsilon$, the translation of A by x is denoted $A_x = \{y \mid \text{for some, } a \in A, y = a + x\}$.

So $B = \{(0, 0), (1, 0)\}$, and therefore $\tilde{B} = \{(0, 0), (-1, 0)\}$. Rather than tediously listing all the 17 elements of A, just draw

$A_{(0,0)} =$ 

, $A_{(-1,0)} =$ 

then

$A \ominus B = A_{(0,0)} \cap A_{(-1,0)} =$ 

.

The second line of Figure 7.1 illustrates the erosion on the right, compared to dilation on the left.

7.2.3 Properties of Dilation and Erosion

Here, we discuss a couple of important properties of the two morphological operators.

- Commutative property: Dilation is commutative, which means we can dilate images in any order without changing the result:

$$A \oplus B = B \oplus A \, . \tag{7.8}$$

- Associative property: Dilation is also associative, which means we can group images in a dilation in any way we choose without changing the result.

$$(A \oplus B) \oplus C = A \oplus (B \oplus C) \, . \tag{7.9}$$

- Distributive property: This property says that in an expression where we need to dilate an image that is the union of two images, we can dilate first and then take the union. In another word, the dilation can be distributed over all the terms inside the parentheses. That is,

$$A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C) \, . \tag{7.10}$$

- Increasing property: If $A \subseteq B$, then

$$A \oplus K \subseteq B \oplus K, \tag{7.11}$$

  for any s.e. K. When this property holds, we say the operator is *increasing*.
- Extensive and anti-extensive properties: If we say an operator is *extensive*, we mean that applying this operator to a set, A, produces an answer that contains A. If the s.e. contains the origin (that is, element (0,0)), dilation is extensive:

$$A \oplus K \supseteq A, \text{ if } (0, 0) \in K \tag{7.12}$$

  As you might guess, erosion has some extensive properties as well: that is, erosion is "anti-extensive," if $(0, 0) \in K$, then $A \ominus K \subseteq A$, where $(0, 0)$ indicates the element at the origin.
- Duality: Duality is similar to DeMorgan's laws: it relates set complement, dilation, and erosion.

$$\begin{aligned} (A \ominus B)^c &= A^c \oplus \tilde{B} \\ (A \oplus B)^c &= A^c \ominus \tilde{B} \end{aligned} \tag{7.13}$$

  where the superscript c denotes set complement.
- Other properties of erosion:

$$A \ominus (B \oplus C) = (A \ominus B) \ominus C \tag{7.14}$$

$$(A \cup B) \ominus C \supseteq (A \ominus C) \cup (B \ominus C) \tag{7.15}$$

$$A \ominus (B \cap C) \supseteq (A \ominus B) \cup (A \ominus C) \tag{7.16}$$

$$A \ominus (B \cup C) = (A \ominus B) \cap (A \ominus C) \, . \tag{7.17}$$

  A cautionary note: You cannot do cancellation with morphological operators because dilation and erosion are not inverses of each other. For example: if $A = (B \ominus C)$, and we dilate both sides by C, we would get $A \oplus C = (B \ominus C) \oplus C$. If dilation and erosion were inverses of each other, the RHS would be just B. However, the RHS is in fact the opening of B by C and not simply B. We will explain the opening operator next.

7.2.4 Opening and Closing

The opening of A by s.e. B is written as

$$A \circ B \equiv (A \ominus B) \oplus B \tag{7.18}$$

Figure 7.3 Demonstration of closing in the inspection of a PC board: TOP: a circuit board conductor with an open circuit due to a manufacturing defect. CENTER: Dilation of the conductor effectively erases the gap. BOTTOM: After erosion, the gap is gone.

and, as you might guess, the closing of A by s.e. B is written as

$$A \bullet B \equiv (A \oplus B) \ominus B \tag{7.19}$$

So what is the purpose of all this? Let's do an example: inspection of printed circuit boards. Figure 7.3(a) shows a picture of a PC board with two traces on it. On one of the traces, a hair was stuck to the board when it went through the wave solder machine, causing a narrow gap in the solder and resulting in an open circuit. We will use closing to identify the gap. First, dilate the image using a small s.e. We choose an s.e. that is smaller than the features of interest (the traces) but larger than the defect. The dilation then looks like Figure 7.3(b). Now, we erode back using the same s.e., and, perhaps to your surprise, the defect is gone. See Figure 7.3(c). For inspection purposes, one could now subtract the original from the closed, and the difference image would contain only the defect. Furthermore, these operations can be done in hardware, blindingly fast.

Here is another example demonstrating some interesting capabilities of combining the opening and closing operators. Figure 7.4 shows an image (of the numeral 2) corrupted by noise in both the foreground area and background area [Figure 7.4(a)]. The size of the noise is 2×2. If we choose a structuring element that is just a little bit bigger than the size of the noise, e.g., 3×3, by applying the opening operator first, the background noise is eliminated [Figure 7.4(c)]; then if we apply the closing operator on the resulting image, the foreground noise is eliminated [Figure 7.4(e)]. So by concatenating the opening and

(a) A **(b)** $A \ominus B$ **(c)** $A \ominus B \oplus B$ or $A \circ B$ **(d)** $(A \circ B) \oplus B$ **(e)** $(A \circ B) \oplus B \ominus B$ or $A \circ B \bullet B$

Figure 7.4 Demonstration of the morphological filter (opening + closing).

closing operators, we obtain a perfectly clean image. We often refer to the operation of "opening + closing" as the *morphological filter*, as it effectively filters out noise in the image.

7.2.5 Properties of Opening and Closing

Some properties of opening and closing are listed below (some you should be prepared to figure out).

- Duality: $(A \circ K)^c = A^c \bullet \tilde{K}$

  Proof of duality: notice how this proof is done. We expect students to do proofs this carefully.

$$\begin{aligned} (A \circ K)^c &= [(A \ominus K) \oplus K]^c && \text{definition of opening} \\ &= (A \ominus K)^c \ominus \tilde{K} && \text{complement of dilation} \\ &= (A^c \oplus \tilde{K}) \ominus \tilde{K} && \text{complement of erosion} \\ &= A^c \bullet \tilde{K} && \text{definition of closing} \end{aligned} \tag{7.20}$$

- Idempotent: Opening and closing are idempotent. That is, repetitions of the same operation have no further effect:

$$A \circ K = (A \circ K) \circ K$$
$$(A \bullet K) = (A \bullet K) \bullet K$$

- Closing is extensive: $A \bullet K \supseteq A$.
- Opening is anti-extensive: $A \circ K \subseteq A$.
- Images dilated by K remain invariant under opening by K. That is, $A \oplus K = (A \oplus K) \circ K$.

  Proof:

1. $A \bullet K \supseteq A$ since closing is extensive
2. $(A \bullet K) \oplus K \supseteq A \oplus K$ dilation is increasing
3. $((A \oplus K) \ominus K) \oplus K \supseteq A \oplus K$ definition of closing
4. $(A \oplus K) \circ K \supseteq A \oplus K$ definition of opening
5. for any B, $B \circ K \subseteq B$ opening is anti-extensive
6. therefore, $(A \oplus K) \circ K \subseteq A \oplus K$ substitution of $A \oplus K$ for B
7. $(A \oplus K) \circ K = A \oplus K$ since $A \oplus K$ is both greater or equal to and less or equal to $(A \oplus K) \circ K$, so only equality can be true.

7.3 Grayscale Morphology

In this section, the concepts of binary morphology are extended to images that are not necessarily binary. We first discuss in section 7.3.1 grayscale morphology with structuring elements that have constant brightness, also called *flat* structuring elements. Then, in section 7.3.2, we consider structuring elements that have a variation in brightness values, also called *nonflat* structuring elements.

$k_1 =$

1		1	
0		2	
-1			
	-1	0	1

, $k_2 =$

-1		6	
0		6	
1			
	-1	0	1

Figure 7.5 The s.e. on the left is not flat. Defining the center, (0, 0), to be the origin, it could be described by the set of triples {(0, 1, 1), (0, 0, 2)}. The s.e. on the right is flat. However, we will see that both can be used as if they were flat by simply ignoring the brightness.

When thinking about grayscale morphology, it is sometimes helpful to think of images as triples, consisting of the x coordinate, the y coordinate, and the brightness. For example, the triple (3, 4, 5) states that the pixel at $x = 3, y = 4$ has brightness 5.

Figure 7.5 illustrates two structuring elements, showing the difference between flat and nonflat structuring elements.

7.3.1 Grayscale Images Using Flat Structuring Elements

A structuring element is defined as *flat* if it has a constant height (usually 1) at every point. Using a flat s.e. provides some simplification that allows the student to make the transition to true grayscale morphology. The basic morphological operations of dilation and erosion are described in this section. Opening and Closing can be determined using Eqs. 7.18 and 7.19, as before.

Because the images are grayscale, it becomes awkward to use set concepts to describe images, and we usually revert to function notation. (Note: it is *possible* to do these operations using set operations, and we describe these in section 7.3.3).

The coordinates of a point in the image will be denoted by a vector $\mathbf{x}$ or similar boldfaced symbol. Using this notation, it becomes easy to think about morphological operations in 3 or higher dimensional spaces. Similarly, the coordinates of a point in the structuring element will be denoted by the vector $\mathbf{s}$.

Now, we need a notation for describing the result of dilation in a functional way. This can easily be accomplished by using the operation as the name of the function. For example, the result of dilating image f by structuring element k, as a function of position $\mathbf{x}$ is simply $(f \oplus k)(\mathbf{x})$. There are other notations in the literature, such as $\delta_x^k(f)$, but we prefer this one.

Grayscale Dilation

The dilation of a grayscale image by a flat s.e. can be defined as

$$(f \oplus k)(\mathbf{x}) = \max_{\mathbf{s} \in k} \{f(\mathbf{x} - \mathbf{s})\} \; . \tag{7.21}$$

Observe that Eq. 7.21 makes no reference to the brightness of the structuring element, only the coordinates. Therefore, if this equation is used, it is the same as having a flat structuring element, and only the shape of the s.e. matters, not its brightness.

Before we continue, look at Eq. 7.21[3] and, as we did in section 7.2, interpret what it means in the context of binary images and binary structuring elements. $\mathbf{s}$ is the coordinate

[3] Remember: $\mathbf{x}$ is a coordinate in the image, $\mathbf{s}$ is a coordinate in the s.e.

$f =$

6					
5					
4		10	20	30	
3			40		
2		50	60	70	
1					
0					
	0	1	2	3	4

, $k =$

1		1	
0		1	
-1			
	-1	0	1

,

$(f \oplus k)(\mathbf{x}) =$

6					
5		10	20	30	
4		10	40	30	
3		50	60	70	
2		50	60	70	
1					
0					
	0	1	2	3	4

Figure 7.6 A grayscale image dilated by a flat s.e.

vector of a particular point in the s.e., defined in the coordinate system of the s.e. For example, the s.e., B, in Eq. 7.2 could be written as the set

$$k(x, y) = \delta_{x,y} + \delta_{x,y-1} \ , \tag{7.22}$$

where δ is the Kronecker delta, equal to one if both subscripts are zero.

Suppose some pixel is located in the image at $\mathbf{x}$. Let $\mathbf{s}$ be a coordinate pair in the s.e. The dilated image at $\mathbf{x}$ should be set to the maximum over all the pixels that overlap the s.e. In the binary case, that maximum will be 1. This is exactly what was done in the last section, but without the function notation.

In Figure 7.6, we give an example of a grayscale image, f, dilated by a flat s.e., k. The numerical values marked on the image indicates the grayscale intensity at that pixel.

Grayscale Erosion

If, as above, we think of dilation in terms of the location of the origin of the s.e., relative to the region being dilated, we can think of erosion in the same way. That is,

$$(f \ominus k)(\mathbf{x}) = \min_{\mathbf{s} \in k} \{f(\mathbf{x} + \mathbf{s})\} \ , \tag{7.23}$$

whereas in Eq. 7.21, $\mathbf{x} - \mathbf{s}$ is used for dilation, a plus sign is used in erosion.

In Figure 7.7, the same grayscale image example as in Figure 7.6 is repeated, but erosion is applied.

In Figure 7.8, grayscale dilation and erosion are applied to a grayscale image. To understand the results, the reader should remember that dilation is defined as dilation of the *bright* areas. Therefore, the dilated face seems slightly larger, whereas erosion of the white area effectively results in dilation of the black areas. Compare the eyes and eyebrows in the eroded and dilated images. The same effect is also demonstrated in Figure 7.9, which is a zoom of the cartoon in Figure 6.5. Again, the erosion of white areas produces apparent dilation of black areas. This effect is responsible for the apparent thickening of the black lines.

$$f = \begin{array}{|c|c|c|c|c|c|}\hline 6 & & & & & \\\hline 5 & & & & & \\\hline 4 & & 10 & 20 & 30 & \\\hline 3 & & & 40 & & \\\hline 2 & & 50 & 60 & 70 & \\\hline 1 & & & & & \\\hline 0 & & & & & \\\hline & 0 & 1 & 2 & 3 & 4 \\\hline\end{array}, \quad k = \begin{array}{|c|c|c|c|}\hline 1 & & 1 & \\\hline 0 & & 1 & \\\hline -1 & & & \\\hline & -1 & 0 & 1 \\\hline\end{array},$$

$$(f \ominus k)(\mathbf{x}) = \begin{array}{|c|c|c|c|c|c|}\hline 6 & & & & & \\\hline 5 & & & & & \\\hline 4 & & & & & \\\hline 3 & & & 20 & & \\\hline 2 & & & 40 & & \\\hline 1 & & & & & \\\hline 0 & & & & & \\\hline & 0 & 1 & 2 & 3 & 4 \\\hline\end{array}$$

Figure 7.7 A grayscale image eroded by a flat s.e.

Figure 7.8 LEFT: Original image. MIDDLE: Grayscale dilation. RIGHT: Grayscale erosion.

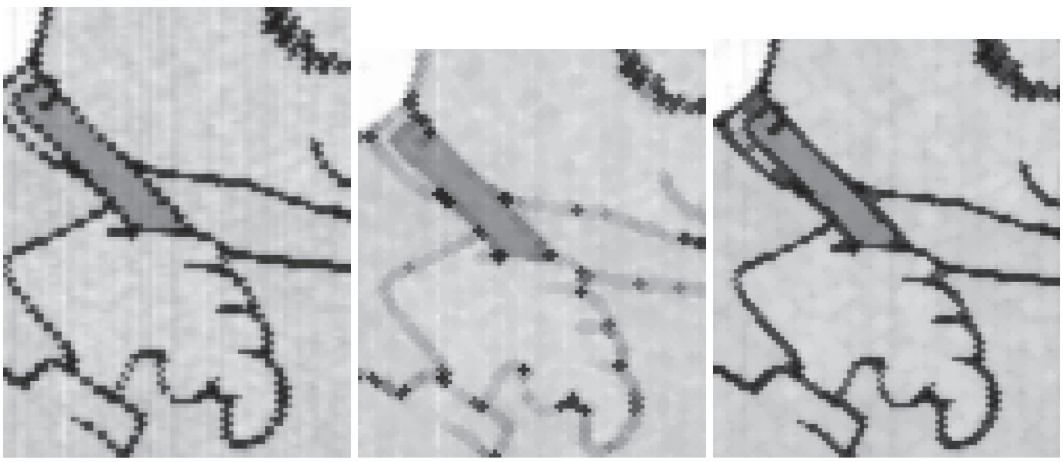


Figure 7.9 LEFT: Original image. MIDDLE: Grayscale dilation. RIGHT: Grayscale erosion.

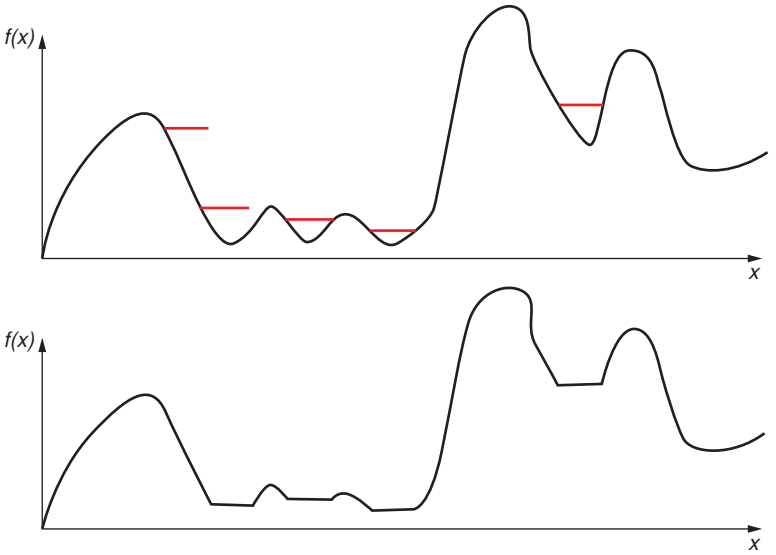


Figure 7.10 The function of grayscale closing to remove noise is illustrated in one dimension. Here, the structuring element is represented by a horizontal line that is placed at every position where it will fit above the brightness function. The function is then not allowed to be lower than the s.e. This process clips off small negative excursions of the signal while leaving the larger portions of the signal unchanged.

Grayscale Opening and Closing

Opening and closing are defined as they were in the previous section, using the results from dilation and erosion.

One of the principal applications of grayscale morphology is to remove noise. Figure 7.10 illustrates what grayscale closing does in one dimension. (The same thing happens in 2D, it's just harder to visualize.) The structuring element may be thought of as a fixed-length horizontal line that moves along the image brightness function seeking lower and lower positions until it hits a spot where it touches the brightness function on both sides. The brightness at which this happens becomes the lowest value of the brightness function, as illustrated in the lower figure.

From the previous discussion, we see that there is no requirement for the region to be binary any more. However, in this formulation, the s.e. must be flat. That requirement is relaxed in the next section.

7.3.2 Grayscale Images Using Grayscale Structuring Elements

In this section, the concept of grayscale morphology is extended to include the use of a nonflat structuring element. Only the equations for dilation and erosion are provided, since everything else can be derived from those.

The dilation of a grayscale image by a grayscale s.e. is accomplished by a very small extension to Eq. 7.21. We allow the s.e. to have nonbinary values.

$$(f \oplus k)(\mathbf{x}) = \max_{\mathbf{s} \in k} [f(\mathbf{x} - \mathbf{s}) + k(\mathbf{s})] \ . \tag{7.24}$$

$f =$

6					
5					
4		10	20	30	
3			40		
2		50	60	70	
1					
0					
	0	1	2	3	4

, $k =$

1		2	
0		5	
-1			
	-1	0	1

,

$(f \oplus k)(\mathbf{x}) =$

6	5	5	5	5	5
5	5	12	22	32	5
4	5	15	42	35	5
3	5	52	62	72	5
2	5	55	65	75	5
1	5	5	5	5	5
0					
	0	1	2	3	4

, $(f \ominus k)(\mathbf{x}) =$

6					
5	-5	-5	-5	-5	-5
4	-5	-2	-2	-2	-5
3	-5	-5	18	-5	-5
2	-5	-2	38	-2	-5
1	-5	-5	-5	-5	-5
0	-5	-5	-5	-5	-5
	0	1	2	3	4

Figure 7.11 Grayscale morphology using a grayscale s.e. Cells are left blank if computing at that point would require nonexistent pixels.

The erosion of a grayscale image by a grayscale s.e. is accomplished by

$$(f \ominus k)(\mathbf{x}) = \min_{\mathbf{s} \in k} [f(\mathbf{x} + \mathbf{s}) - k(\mathbf{s})] \ . \tag{7.25}$$

As before, the reverse of the structuring element is avoided by using a "$-$" in $\mathbf{x} - \mathbf{s}$ in the dilation and a "$+$" in $\mathbf{x} + \mathbf{s}$ in the erosion.

Figure 7.11 shows an example of dilation and erosion using the same grayscale image as in Figure 7.6 but a grayscale s.e. For example, when $\mathbf{x} = (1, 4)$,

$$\begin{aligned} f \oplus k)(\mathbf{x}) &= \max\{f((1,4) - (0,0)) + k(0,0), f((1,4) - (0,1)) + k(0,1)\} \\ &= \max\{15, 2\} = 15; \end{aligned}$$

similarly,

$$\begin{aligned} f \ominus k)(\mathbf{x}) &= \min\{f((1,4) + (0,0)) - k(0,0), f((1,4) + (0,1)) - k(0,1)\} \\ &= \min\{10, -2\} = -2. \end{aligned}$$

Figure 7.11 illustrates one reason for the fact that grayscale structuring elements are seldom used – they can produce negative values. Of course, these are easily dealt with by, for example, setting negative pixels to zero. Still, it is easier to understand and design flat structuring elements.

There is one other variation that could be added to grayscale morphology at this point: allow the coordinates $\mathbf{x}$ and $\mathbf{s}$ to be continuous. That variation is not pursued in this introductory book.

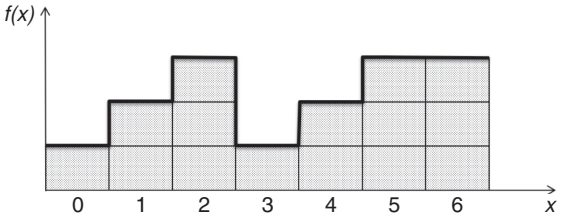


Figure 7.12 An illustration of umbra. The dark line is the TOP of the umbra, and is the brightness function.

7.3.3 Grayscale Morphology Using Set Operations

While the definitions in sections 7.3.1–7.3.2 are straightforward and easy to implement, another way to define grayscale morphological operations allows the use of the binary morphology operations. To accomplish that, a new concept, the umbra, is defined:

The *umbra*, $U(f)$, of a two-dimensional grayscale image f is the set of all ordered triples, (x, y, U), which satisfy $0 < U \leq f(x, y)$. If we think of f as being continuously valued, then $U(f)$ is an infinite set. To make morphological operations feasible, f is assumed to be quantized to M values.

To illustrate the concept of umbra, let f be one-dimensional and the pixel value at the corresponding coordinate is

$$f(x) = [1, 2, 3, 1, 2, 3, 3] .$$

f is shown as a brightness function in Figure 7.12.

In Figure 7.12, the heavy black line represents f, and the umbra is the shaded area under f. Following this figure, the heavy black line is the TOP of the umbra. The umbra is a set of ordered pairs:

$$\begin{aligned} U(f) = \{&(0, 1), (1, 1), (1, 2), (2, 1), (2, 2), (2, 3), (3, 1), (4, 1), (4, 2), \\ &(5, 1), (5, 2), (5, 3), (6, 1), (6, 2), (6, 3)\}. \end{aligned}$$

Here's the trick: Although the gray-level image is no longer binary (and therefore not representable by set membership), the umbra does have the properties of a set. Therefore, it is possible to define the dilation of a grayscale image f by a grayscale s.e., k, using the binary dilation operation as

$$f(x, y) \oplus k(x, y) = TOP(U(f) \oplus U(k)) \tag{7.26}$$

and erosion is similarly defined. Furthermore, grayscale opening and closing can be defined in terms of grayscale dilation and erosion.

Generalizing this concept to two-dimensional images, the umbra becomes three-dimensional, a set of triples

$$U(f(x, y)) = \{(x, y, z) \mid (z \leq f(x, y))\} . \tag{7.27}$$

2	1	1	1	1	■	1	2	3	4	5
1	■	■	■	■	1	■	1	2	3	4
1	■	1	1	1	2	1	■	1	2	3
2	1	■	1	2	3	2	1	■	1	2
2	1	■	1	2	3	2	1	■	1	2
3	2	1	■	1	2	1	■	1	2	3
3	2	1	■	1	2	2	1	■	1	2
3	2	1	■	1	1	1	1	■	1	2
2	1	■	■	■	■	■	■	■	1	2
3	2	1	1	1	1	1	1	1	2	3

Figure 7.13 The boundary of a region is shown in black. One variation of the DT of the region, computed using a 4-connected definition, is shown.

Then, grayscale dilation and erosion may be written compactly as

$$\begin{aligned} f(x,y) \oplus k(x,y) &= \{(x,y,z) \mid (z \leq \max(f(x-x_1, y-y_1) + k(x_1,y_1)))\} \ \forall (x_1,y_1) \\ f(x,y) \ominus k(x,y) &= \{(x,y,z) \mid (z \leq \min(f(x+x_1, y+y_1) - k(x_1,y_1)))\} \ \forall (x_1,y_1) \end{aligned} \tag{7.28}$$

for $(x_1, y_1) \in \Omega \subset Z \times Z$, where Ω denotes the set of possible pixel locations, assumed here to be positive and integer.

7.4 The Distance Transform

A very important application of morphological operations is the distance transform (DT). The DT may be defined in several ways, but we will present a simple one here. The distance transform is an iconic representation of an image, in which each pixel in the DT contains the distance of the corresponding pixel in the input image from some feature. Most often, the feature is an edge. In this case, the distance from a pixel to an edge is defined as the distance to the closest edge point. In this section, the boundary of region R is denoted by the set of points ∂R. Figure 7.13 illustrates the DT of both the interior and exterior of a region whose boundary is known.

Depending on the distance measure used, there are different versions of the DT. The Euclidean DT is described by

$$DT(\mathbf{x}) = \min_{\mathbf{y} \in \partial R} \| \mathbf{x} - \mathbf{y} \| \tag{7.29}$$

where $\mathbf{x}$ and $\mathbf{y}$ are 2-vectors of coordinates.

The distance transform that results from some DT algorithm is dependent on the definition of distance, of neighbor, and the details of the algorithm. Clearly, the transform shown in Figure 7.13 does not provide the Euclidean distance from each pixel to the nearest boundary point.

7.4.1 Computing the DT Using Iterative Nearest Neighbor

To start off, the initial condition of the iterative process is that $DT(\mathbf{x}) = 0$ for $\mathbf{x} \in \partial R$. That is, the DT at the boundary pixels of the region is set to 0. Then at iteration 1, look at each pixel and determine if it has a neighbor that has a label of 0. If so, label it with a 1. Do

$$T = \begin{array}{|c|c|}\hline & 1 \\ \hline 1 & +0 \\ \hline \end{array}$$

Figure 7.14 Mask used to calculate the 4-connected DT. The origin of the mask is denoted by a plus sign.

that for all the pixels, and then repeat, looking for unlabeled pixels with labeled neighbors. Denoting $\eta(\mathbf{s})$ as the set of neighbors of a pixel, we end up with a general algorithm:

At iteration i, for every unlabeled point $\mathbf{x}$ that has at least one labeled neighbor $\mathbf{y}$, $DT(\mathbf{x}) = \min_{y \in \eta(\mathbf{x})} DT(y) + 1$. Repeat this until all the DT points have a label.

Using this algorithm with a 4-connected definition of neighborhood produces DT in which the distance from a point to the boundary is the *Manhattan distance*.

7.4.2 Computing the DT Using Binary Morphological Operators

One may estimate the DT using conventional morphological operations: Let us suppose we want the distance from a point $\mathbf{x}$ to the outside edge of an object. To accomplish that, repetitively erode the image, using some "appropriate" structuring element (usually round and centered at zero). Each time a pixel disappears, record the iteration at which that pixel vanished. Store the iteration number in the corresponding pixel of the DT. Pretty simple, isn't it? That method does not quite give the Euclidean distance, but it serves well for many purposes. Huang and Mitchell [7.3] showed how to obtain the Euclidean distance transform using grayscale morphology, and Breu et. al [7.2] showed how to compute the Euclidean DT in linear time.

For a structuring element that is strictly convex, the points go in the direction of the normal to the boundary of the region being eroded [7.6].

7.4.3 Computing the DT Using a Mask

Computing the distance transform may be accomplished by iterative application of a mask, such as the one illustrated in Figure 7.14.

At iteration m, the distance transform is updated using the following equation,

$$D^m(x, y) = \min_{k,l \in T}(D^{m-1}(x + k, y + l) + T(k, l)) \; . \tag{7.30}$$

A more detailed explanation follows: first, the distance transform, $D(x, y)$, is initialized to a special symbol denoting "infinity" at every non-edge point, $D^0(x, y) = \infty$, $\forall (x, y) \notin \partial R$, and to zero at every edge point $D^0(x, y) = 0$, $\forall (x, y) \in \partial R$. Then, application of the mask starts at the upper left corner of the image, placing the origin of the mask over the (1,1) pixel of the image, and applying Eq. 7.30 to calculate a new value for the DT at (1,1). In the example of Figure 7.15, the DT is illustrated with infinities for non-edge points, and zeros for edge points. The origin of the mask in Figure 7.14 is applied (or overlaid) on the pixel with shaded area. The application of Eq. 7.30 produces $\min(1 + 0, 1 + \infty)$ for the distance transform value in the pixel with the shaded area.

0	∞	∞	∞	∞
0	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞

Figure 7.15 Calculation of the distance transform value at the point (1,1) in an image whose upper left point is the point (0,0). The result is 1.

After one pass, top-to-bottom, left-to-right, the mask is reversed (in both directions), and applied again, bottom-to-top, right-to-left.

It is important to emphasize that in the application of a mask for computing the distance transform using Eq. 7.30, the input image is modified by the application, as the application runs. This is called an *in-place* algorithm. An algorithm of this type is normally a bad idea, as it may lead to unexpected results. For example, consider the use of this idea in edge detection. The output of the algorithm, the gradient magnitude, would be written into the image at one pixel, and then become an input pixel in the neighborhood of the next input pixel. However, if properly designed (e.g., Eq. 7.30), this approach can produce correct results much faster than iterative algorithms that write the entire image at each iteration.

Masks other than that of Figure 7.14 produce other variations of the DT. In particular, Figure 7.16 produces the *chamfer map*. If divided by 3, the chamfer map produces a DT that is not a bad approximation to the Euclidean distance.

Figure 7.17 illustrates a boundary and its distance transform. It is difficult to see detail in the grayscale version of the DT, except to observe that the image gets brighter as one moves away from the boundary pixels. However, by randomly assigning colors to the distances, the contours of equal distance become easy to identify, as illustrated in Figure 7.18.

7.4.4 Computing the DT Using the Voronoi Diagram

In later sections of this book, we will be concerned with the junctions between regions and the relationships between adjacent regions. We will occasionally need to consider relationships between regions that do not actually touch. For this, the concept of the Voronoi diagram will be useful. We introduce it here because of the similarities it shares with the distance transform.

Consider the image illustrated in Figure 7.19. In that image, several regions are indicated in gray, inside the white circle. For any region, i, the *Voronoi domain* of that region is the set of points such that those points are closer to points in that region than to points in any other region:

$$V_i = \{\mathbf{x} \mid d(\mathbf{x}, \mathbf{p}_i) < d(\mathbf{x}, \mathbf{p}_j), \forall (j \neq i)\} \tag{7.31}$$

4	3	4
3	+0	

Figure 7.16 A mask whose application produces the (roughly Euclidean) chamfer map.

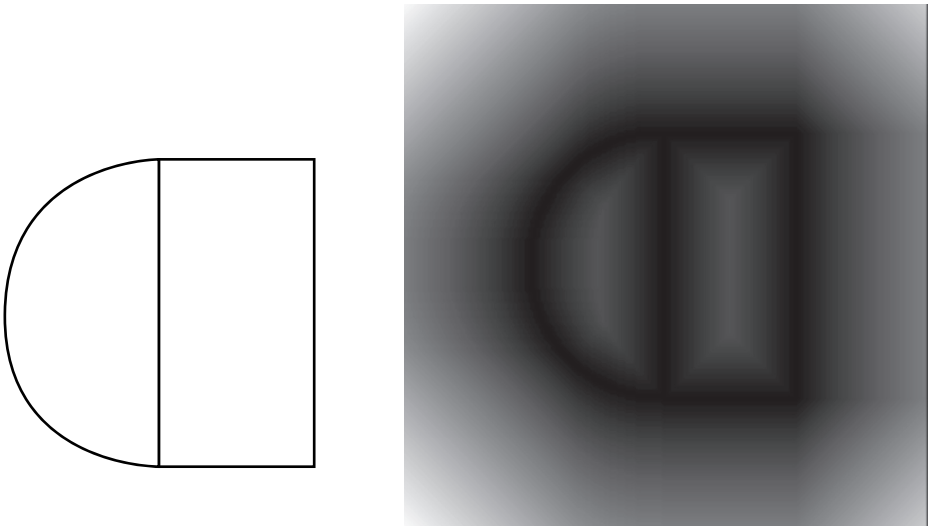


Figure 7.17 LEFT: An edge image. RIGHT: The DT of the LEFT image, computed using the mask of Figure 7.14.

where $\mathbf{p}_i$ denotes the coordinate of any point in region i, d calculates the distance between point $\mathbf{x}$ and point $\mathbf{p}_i$. The set of points equidistant from two of those regions is referred to as the *Voronoi diagram*, and is illustrated by the dark lines in the figure.

Breu et al. showed in [7.2] how the DT of d-dimensional binary image can be computed by constructing and regularly sampling the Voronoi diagram.

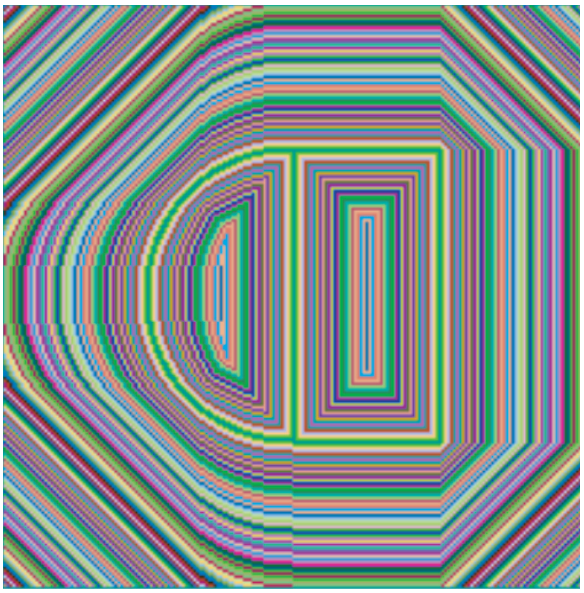


Figure 7.18 The distance transform shown in Figure 7.17 (RIGHT), using a random pseudocolor assignment.

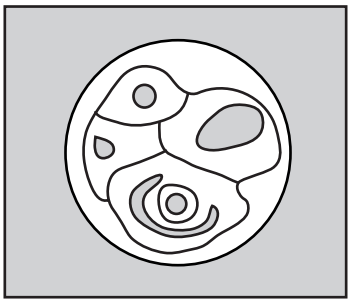


Figure 7.19 Several regions and the resulting Voronoi diagram. Redrawn from Tagare et al. [7.5].

7.5 Applications in Edge Linking

This section presents an application, closing gaps in edges (or edge linking), which combines the use of the distance transform with dilation and erosion to illustrate the power of such methods.

In section 7.2.4, you learned that closing could sometimes be used to close gaps. That suggests a simple algorithm:

- Dilate the boundary pixels with a structuring element on the order of the size of the gap.
- Erode the resulting close boundary by the same s.e.

Unfortunately, this won't work very well in real life. The erosion process, in general, does not know precisely when to stop, and when erosion runs too long, it may create new gaps, defeating the purpose.

There is another method, using *thinning*. Thinning algorithms perform operations similar to erosion [7.1] but do not break connectivity (create gaps). The resulting collection of points is similar to a *skeleton*, which we will discuss in section 10.4.5. The weakness of the thinning algorithms mentioned above is that they do not remember where the boundary was originally. In the following few paragraphs, we show how to use the Distance Transform to accomplish that, and will compare the results with thinning.

In a 2D image, the outer boundary of a region should be a closed curve, but such boundaries often have gaps, due to noise and blur, as illustrated in Figure 7.20. To address this problem, first we compute the DT of the edge image. Next, we replace any point

2	3	3	4	5	6	5	4	4	4
1	2	2	3	4	5	4	3	3	3
	1	1	2	3	4	3	2	2	2
1			1	2	3	2	1	1	1
2	1	1		1	2	1			
3	2	2	1	2	3	2	1	1	1
4	3	3	2	3	4	3	2	2	2
5	4	4	3	4	5	4	3	3	3

Figure 7.20 A boundary with a gap, and its distance transform. Redrawn from [7.4], used with permission

Q	3	3	4	5	6	5	4	4	4
Q	Q	Q	3	4	5	4	3	3	3
■	Q	Q	Q	3	4	3	Q	Q	Q
Q	■	■	Q	Q	3	Q	Q	Q	Q
Q	Q	Q	■	Q	Q	Q	■	■	■
3	Q	Q	Q	Q	3	Q	Q	Q	Q
4	3	3	Q	3	4	3	Q	Q	Q
5	4	4	3	4	5	4	3	3	3

Figure 7.21 A boundary with a gap, and its distance transform, with the DT values less than 3 replaced by Q.

whose DT value is less than a parameter, m, with a special designation Q as illustrated in Figure 7.21.

Now, observe that all the points marked with a Q actually form a dilation of the original boundary. We call this a *DT-guided dilation*, or simply *DT-dilation*.

Next, we illustrate how this use of DT-guided dilation can create closed boundaries. Figure 7.22 illustrates the results of edge detection on an image with two regions. Noise has resulted in a large number of "dropouts," missing points in the edge. In Figure 7.23, we illustrate first the DT-dilation of the previous figure, but with shades of gray denoting the DT value at that point. On the right of that figure, the DT pixels with 8-connected DT values of 2 or less are set to zero, creating a wide, dilated version of the original boundary. The two regions are now separated by this wide boundary.

After DT-dilation, *connected component labeling* is run on the background images. Connected component labeling will be described in detail in section 8.4, but basically, it identifies pixels belonging to each background region separated by the DT-dilated boundary, and marks those pixels differently according to the region to which they belong. The result of running connected components on the DT-dilated image is shown in Figure 7.24. The pixels are said to be *labeled* as belonging to region x or region y.

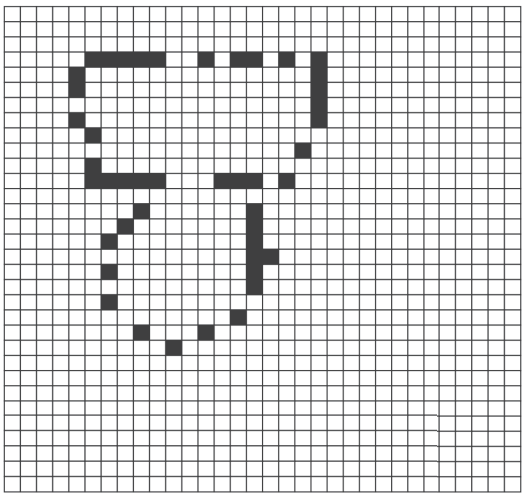


Figure 7.22 Edges of two adjacent regions are corrupted by dropouts. Redrawn from [7.4]. Used with permission.

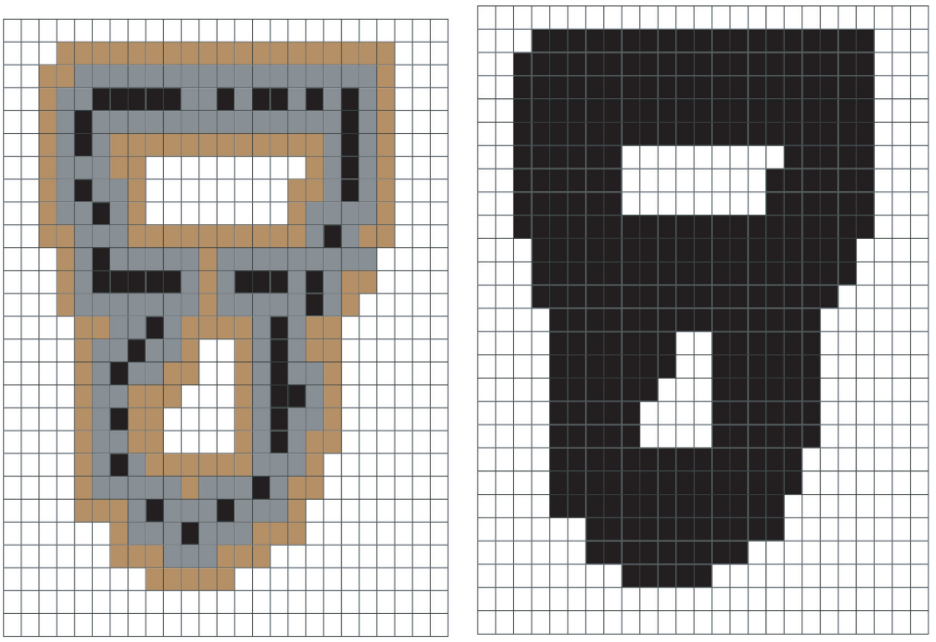


Figure 7.23 LEFT: The DT-dilation of Figure 7.22, using DT distance of 2 or less. Notice that it produces a closed boundary. Original boundary pixels are black. Those with a DT distance of 1 or 2 are successively lighter. RIGHT: The same dilation with all the pixels in the DT having distance of 2 or less set to black. Redrawn from [7.4]. Used with permission.

Now we have dilated the boundary in such a way that the dilated boundary separates two regions. The next step is to get rid of all those black pixels. We accomplish that by an erosion process, assigning black pixels, using the distance transform. We denote this process as *DT-guided erosion*, or simply *DT-erosion*. This erosion is also a simple process:

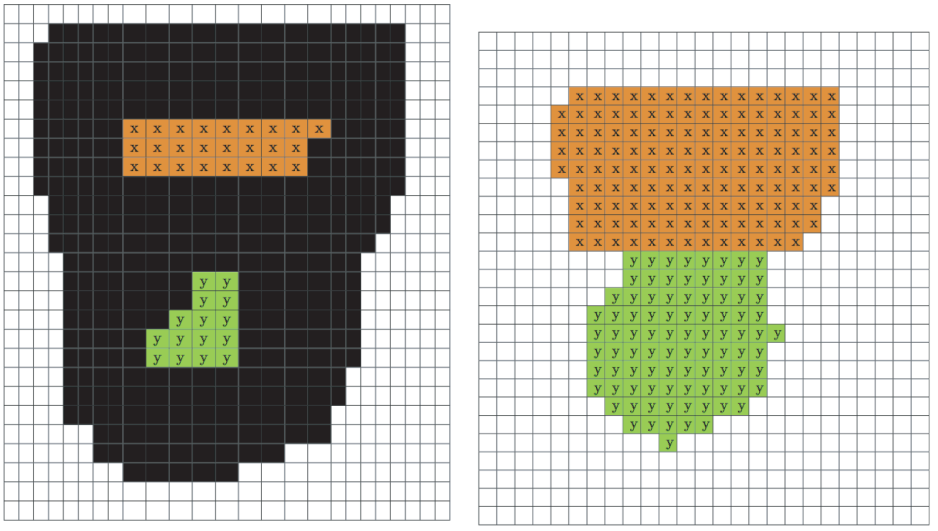


Figure 7.24 LEFT: Running connected components on the background allows regions of the background that are not connected to be distinguished. RIGHT: After DT-erosion, all the boundary pixels are assigned to the closest region. Redrawn from [7.4]. Used with permission.

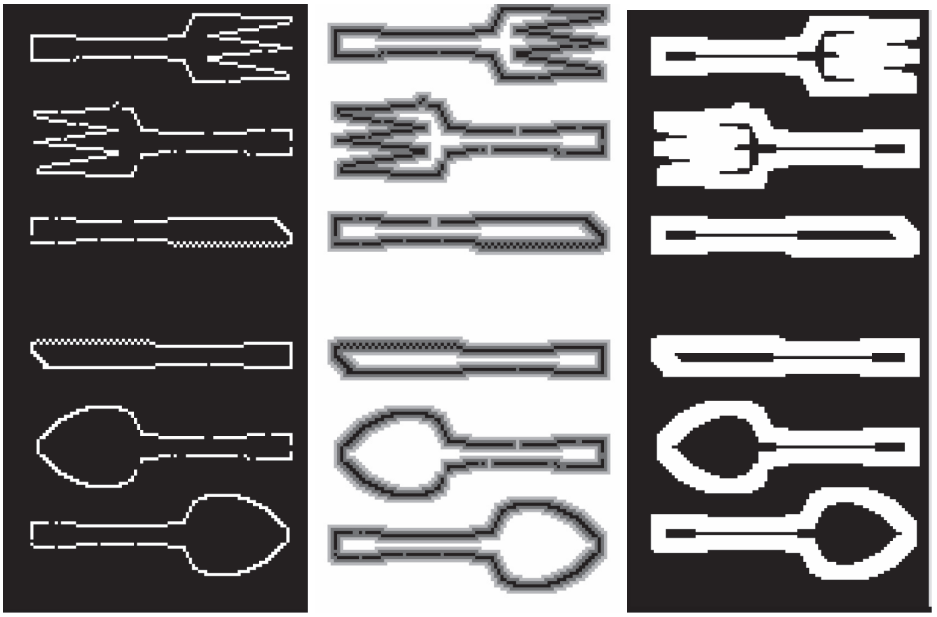


Figure 7.25 LEFT: Output of a noisy boundary detector. (Library Name: cutleryNoisy.png) MIDDLE: Distance transform. (Library Name: cutleryDT.png) RIGHT: Dilated image. (Library Name: cutleryThick.png). Redrawn from [7.4]. Used with permission.

For every pixel with a DT value of 2 (for this example), we assign that pixel to the label of the majority of its neighbors. If ties occur, random decisions are made. After eroding all the black pixels with labels of 2, the process is repeated with labels of 1 and then zero. Finally the labeling on the right of Figure 7.24 results. There is always a problem with boundary labeling: Should one count the boundary pixels themselves as part of the region? This labeling has not only separated the two regions; it has labeled the boundary pixels explicitly as pixels of the region.

In Figure 7.25, we show another example of how DT-dilation and DT-erosion help close the gaps in an image of cutlery. On the left of Figure 7.25, a single image is shown, resulting from edge detection and thresholding on an image of cutlery. The dilation shown on the right is the result of DT-dilation, but it could easily have resulted from conventional binary dilation. One could, at this point, choose to use thinning to find the skeleton of the boundary. The results of thinning are shown in the center of Figure 7.26. One observes that sometimes the thinning works well. But when the relative resolution (the ratio of the spacing of the fork tines to the pixel size) is small, thinning produces the wrong skeleton. This occurs because there is no information in the dilated fork to tell the thinning algorithm how to move. Using DT-erosion allows the boundary to be eroded down to the proper, closed boundary on the right of Figure 7.26.

7.6 Conclusion

In this chapter, we have looked at a particular approach to processing the shape of regions. Morphological operators are useful in both binary images, and in grayscale images. The

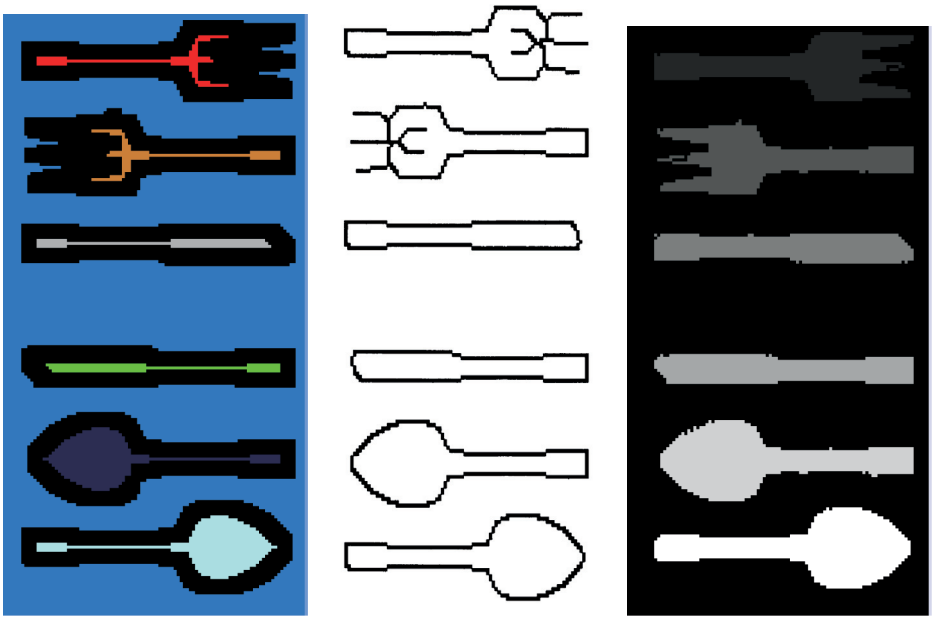


Figure 7.26 LEFT: The result of applying Connected Components to the dilated image. MIDDLE: Result of using thinning to find the closed boundaries. RIGHT: Result of using DT-erosion to find the boundaries. From [7.4]. Used with permission.

distance transform has been the subject of many papers (e.g., [7.2]) that improve the accuracy and/or speed up DT algorithms.

With proper combinations of the morphological operators, we can achieve special effects like noise removal and edge linking in a very efficient manner. However, the authors would like to point out that morphological-based noise removal or edge linking only removes noise features of small scale (i.e., smaller than the radius of the s.e.) and closes gaps of relatively small size. For example, in Figure 7.22, the largest gap in the edge is of 3 pixels. Therefore, a DT distance of 2 or less would work. However, suppose the size of the gap is of 7 pixels. Then the DT-dilation with DT distance of 4 or less would be needed, which would completely close up the background region marked with "x" in Figures 7.23 (LEFT) and 7.24 (LEFT). When there are large gaps in the edge, we can make use of the Hough transform, a technique to be discussed in detail in Chapter 9.

7.7 Assignments

Assignment 7.1: In section 7.2.3, we stated that dilation is commutative because addition is commutative. Erosion also involves addition, but one of the two images is reversed. Is erosion commutative? Prove or disprove it.

Assignment 7.2: Prove (or disprove) that binary images eroded by a kernel, K, remain invariant under closing by K. That is, prove that $A \ominus K = (A \ominus K) \bullet K$.

Assignment 7.3: Show that dilation is not necessarily extensive if the origin is not in the s.e.

Assignment 7.4: Prove that dilation is increasing.

Assignment 7.5: Let C be the class of binary images that have only **one** dark pixel. For a particular image, let that pixel be located at (i_0, j_0).

Using erosion and dilation by kernels that have (0, 0) as an element, devise an operator, that is, a set of erosions and dilations, and structuring elements (you may need only one s.e. or you may need more than one), which, when applied to an element of C, would output an image with the dark pixel shifted to $(i_0 + 2, j_0 + 1)$. Disregard boundaries.

Assignment 7.6: Which of the following statements is correct? (You should be able to reason through this, without doing a proof.)

$$\begin{aligned} &1)\ (A \ominus B) \ominus C = A \ominus (B \ominus C) \\ &2)\ (A \ominus B) \ominus C = A \ominus (B \oplus C) \end{aligned} \tag{7.32}$$

Assignment 7.7: Use the thresholded images you created in Assignment 5.4 or (it's up to the instructor) Assignment 5.5. Choose a suitable structuring element, and apply opening to remove the noise.

Assignment 7.8: In section 7.4, a mask (Figure 7.16) is given and it is stated that application of this mask produces a distance transform that is "not a bad approximation" to the Euclidean distance from the interior point to the nearest edge point. How bad is it? Apply the algorithm to the following image:

$$\begin{matrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & 0 & 0 & \infty \\ \infty & \infty & \infty & \infty \end{matrix}$$

Your instructor may ask you to do this by hand. Show the output after each pass through the program, or your instructor may ask you to write a program.

CAUTION: If you do this manually, it can be tedious and time-consuming, as you will need to compute by hand a number of steps (something like fifteen). Or, if you write a program: That can be time-consuming too. Be sure you allocate sufficient time.

Assignment 7.9: Consider a region with an area of 500 pixels of which 120 pixels are on the boundary. You need to find the distance transform from each pixel in the interior to the boundary, using the Euclidean distance. (Note: Pixels *on* the boundary are not considered *in* the region – at least in this problem). What is the computational complexity? (Note: You may come up with some algorithm more clever than that used to produce any of the answers below, so if your algorithm does not produce one of these answers, explain it.)

a) 60,000
b) 120,000
c) 45,600
d) 91,200

Assignment 7.10: Trick question: For Assignment 7.9, how many square roots must you calculate to determine this distance transform? Remember, this is the Euclidean distance.

Assignment 7.11: What is the major difference between the output of a thinning algorithm and the maxima of the distance transform? (choose the best answer).

a) A thinning algorithm preserves connectivity. The maxima of the DT are not necessarily connected.

b) The maxima of the DT are unique, thinning algorithms do not produce unique results.

c) The DT preserves connectivity. Thinning algorithms do not.

d) Thinning algorithms produce the intensity axis of symmetry. The DT does not.

Bibliography

[7.1] C. Arcelli, L.. Cordella, L., and S. Levialdi. Parallel thinning of binary pictures. *Electron. Letters*, (11): 1975.

[7.2] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman. Linear time Euclidian distance transform algorithms. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(5), 1995.

[7.3] C. Huang and O. Mitchell. A euclidian distance transform using grayscale morphology decomposition. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(4), 1994.

[7.4] W. Snyder, M. Hsiao, K. Boone, M. Hudacko, and B. Groshong. Closing gaps in edges and surfaces. *Image and Vision Computing*, October 1992.

[7.5] H. Tagare, F. Vos, C. Jaffe, and J. Duncan. Arrangement: A spatial relation between parts for evaluating similarity of tomographic section. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(9), 1995.

[7.6] R. van den Boomgaard and A. Smeulders. The morphological structure of images: The differential equations of morphological scale-space. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(11), 1994.

Part III

Image Understanding

Finally! You have an image with relatively sharp edges (except for the occasional missing edge pixel and other defects), smooth region interiors (except that some variation still exists in brightness), and smooth background (except that there may be other objects in the background). Now, all you have to do is identify specific regions (segmentation), describe those regions (measure shapes and calculate features), and recognize them (match them to models). As you can guess, it's not quite that simple. Nothing is ever that simple. Here's a simple picture. Can your computer software recognize it?

This image has library name: StingRay.jpg.

8 Segmentation

The partition between the sage and the fool is more slender than the spider web.
– Kahlil Gibran

8.1 Introduction

Segmentation is the process of separating objects from background. It is the foundation for all the subsequent processes like shape analysis and object recognition.

A *segmentation* of a picture is a partitioning into connected regions, where each region is homogeneous in some sense and is identified by a unique label. For example, in Figure 8.2 (a "label image"), region 1 is identified as the background. Although region 4 is also background, it is labeled as a separate region, since it is not connected to region 1.

The term "homogeneous" deserves some discussion. It could mean all the pixels are of the same brightness, but that criterion is too strong for most practical applications. It could mean that all pixels are close to some representative (mean) brightness. Stated more formally [8.56], a region is homogeneous if the brightness values are consistent with having been generated by a particular probability distribution. In the case of range imagery [8.31] where we (might) have an equation that describes the surface, we could say a region is homogeneous if it can be described by the combination of that equation and some probabilistic deformation. For example, if all the points in a region of a range image lie in the same plane except for deviations whose distance from the plane may be described by a particular Gaussian distribution, one could say this region is homogeneous.

In this chapter, several ways to perform segmentation are discussed. We progress through problems and methods of increasing complexity:

- (Section 8.2) Threshold-based techniques are guaranteed to form closed regions because they simply assign all pixels above (or below, depending on the problem) a specified threshold to be in the same region. However, using a single threshold only allows regions to be classified as "foreground" and "background."
- (Section 8.3) Brightness segmentation into two classes is the simplest case. It becomes more challenging when color images are considered and similarities of colors must be considered.
- (Section 8.4) Another level of complexity occurs when it is necessary to determine whether two apparently separate regions are really one region. That is, do they touch? For this, we need connected components.

Figure 8.1 An image with two foreground regions.

- (Section 8.5) In many applications, the segmentations returned from the above methods are inadequate because they have corrupted and/or incomplete boundaries. For those problems, contour-based methods work best.
- (Section 8.6) For some problems, it is best to think of the brightness function as dark valleys surrounded by bright mountains, and in such problems, watershed methods work well [8.4]. A watershed method consists of flooding the image with (by analogy) water, in which region boundaries are erected to prevent water from different basins from mixing.
- (Section 8.7) Optimization may be directly applied, treating the image as a graph with pixels as nodes and edges between neighboring pixels. Optimal cuts in the image graph provide a means to define and optimize the quality of the segmentation. These methods can be computationally intensive but provide excellent performance if it is possible to derive an objective function appropriate for the problem at hand.
- (Section 8.8) The mean field annealing algorithm (used for noise removal in Chapter 6) may be applied to obtain an optimal piecewise-constant image, which leads directly to a segmentation.

Before continuing, note that segmentation software can make mistakes, as will be discussed in more detail in section 8.9. For that reason, it is necessary to distinguish between the "correct" partition of the image and the one returned by the segmenter, which is not necessarily the same. We will refer to the individual components of the ideal image, the

```
1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 2 2 2 2 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1
1 1 1 1 2 2 2 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1
1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 1 1 1 1 1
1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 1 1 1 1
1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 1 1 1
1 1 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 1 1 1
1 1 2 2 4 4 2 2 2 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 1 1 1 1
1 1 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 1 1 1 1 1 1
1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Figure 8.2 A segmentation and labeling of the image in Figure 8.1.

.013

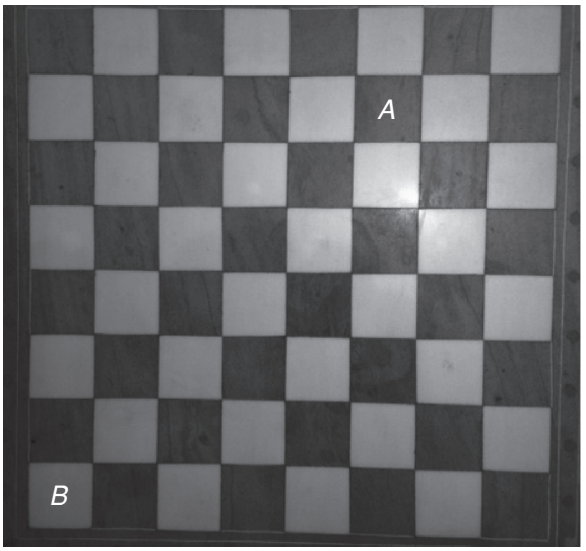


Figure 8.3 In this non-uniformly illuminated picture of a checkboard, the "black" square labeled *A* is actually brighter than the "white" square at *B*, yet humans have no difficulty distinguishing them properly.

model, as *regions*. On the other hand, the components returned by the segmenter will be called *patches*. With skill and care you will always write segmenters that return patches that are identical to the corresponding regions.

8.2 Thresholding: Segmentation Based on Brightness Alone

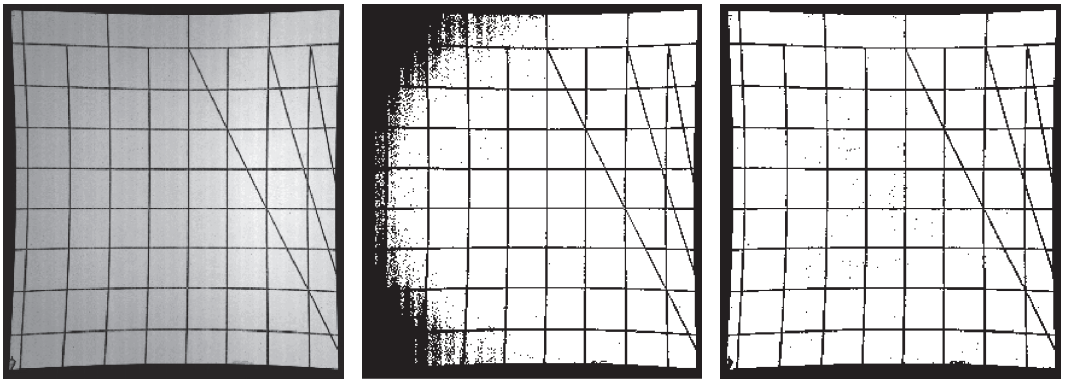
In this section, you will learn how, using brightness alone, to partition an image into regions. You will learn that there are several ways to do this, and see one example using minimization to fit two or more Gaussians to a histogram.

In applications where specific gray values of regions are not important, one can segment a picture into "foreground" (or "objects") and "background" by simply choosing a threshold in brightness. Any region whose brightness is above the threshold is denoted as a foreground *object* and all regions below the threshold are denoted as background (or, depending on the problem, vice versa).

Ways to choose thresholds range from the trivially simple to the very sophisticated. As the sophistication of the technique increases, performance improves, but at the cost of increased computation.

8.2.1 The Local Nature of Thresholding

A single threshold is almost never satisfactory for an entire image. It is nearly always the local contrast between object and background that contains the relevant information. Since camera sensitivity frequently drops off from the center of the picture to the edges due to effects such as lens vignetting or nonuniform illumination as shown in Figure 8.3, it may



(a) Original image. (b) Image segmented using global thresholding. (c) Image segmented using local thresholding.

Figure 8.4 Using the simple thresholding strategy to segment a calibration image with both radiometric and geometric distortions.

be futile to attempt to establish a global threshold. A dramatic example of this effect can be seen in an image of a rectilinear grid, in which the "uniform" white varies significantly over the surface.

Effects such as vignetting are quite predictable and easy to correct. It is more difficult, however, to predict and correct effects of nonuniform ambient illumination, such as sunlight through a window, which changes radically over the day. Since a single threshold cannot provide acceptable performance, we must choose local thresholds. The most common approach is called *block thresholding*, in which the picture is partitioned into rectangular blocks and different thresholds are used on each block. Typical block sizes are 32×32 or 64×64 for 512×512 images. The block is first analyzed and a threshold is chosen; then that block of the image is thresholded using the results of the analysis. In more sophisticated (but slower) versions of block thresholding, the block is analyzed and the threshold computed. Then that threshold is applied only to the single pixel at the center of the block. The block is then moved over one pixel, and the process is repeated.

The simplest strategy for choosing a threshold is to average the intensity over the block and choose $i_{avg} + \Delta i$ as the threshold, where Δi is some small increment, such as 5 out of 256 gray levels. Such a simple thresholding scheme can have surprisingly good results. Figure 8.4 shows (1) a calibration image acquired by a scintillator detecting x-rays, (2) the segmented image using this simple thresholding method, but with the same threshold applied on the entire image (i.e., global thresholding), and (3) the same method applied locally (i.e., local thresholding). We notice the artifact with global thresholding at the left of the image, which is well handled by local thresholding.

8.2.2 Choosing a Threshold through Histogram Analysis

When the simpler schemes fail, one is forced to move to more sophisticated techniques, such as thresholding based on histogram analysis. Before describing this technique, we first define a histogram:

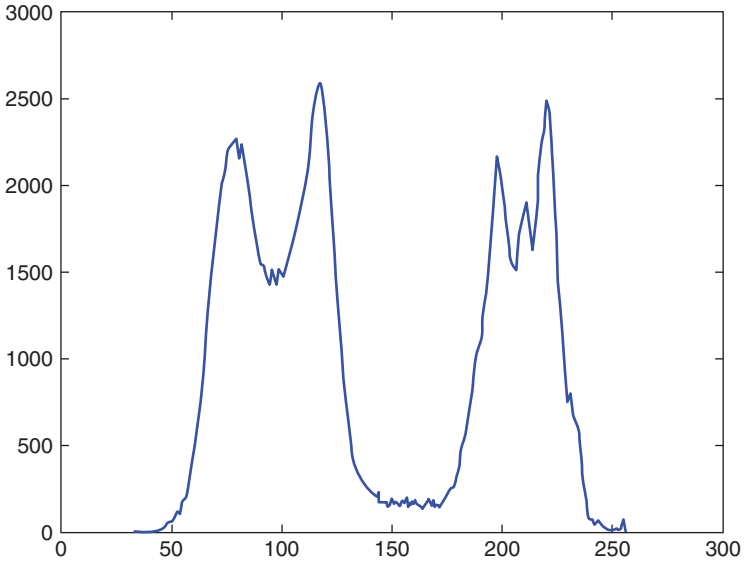


Figure 8.5 A histogram of bimodal image, with many bright pixels (intensity around 200) and many dark pixels (intensity around 100).

The *histogram* $h(f)$ of an image $f(x, y)$ is a function of the permissible intensity values. In a typical imaging system, intensity takes on values between zero (black) and 255 (white). A graph that shows, for each gray level, the number of times that level occurs in the image is called the histogram of the image. Illustrated in Figure 8.5 is a histogram of a checkerboard under reasonably uniform illumination.

Often, instead of using individual brightness values that answer the question "How many pixels have brightness 35?", a histogram is set up to answer the question "How many pixels have brightness between 35 and 55?" In this case, the horizontal axis of the histogram is broken into segments called *bins*. There might, for example, be only 64 bins in an image with brightness ranging from 0 to 255, and all brightnesses in one bin (say, 33 to 36) would be considered the same. Modifying a histogram in this way is called *rebinning*, and is needed when the number of samples is small. For example, suppose you wanted a histogram of the grades in the Computer Vision class, but there are only 30 students. Assuming all the grades are between 50 and 100, it is likely that most bins will contain either zero or one sample. To get a useful histogram, we could break the 50–100 range into ten bins of five points each. With only ten bins, distinct peaks might be visible. If not, then another rebinning might be required.

If the histogram $h(f)$ is divided by the number of pixels in the image, we arrive at a function that has all the properties of a discrete probability. Henceforth, when we use the notation $h(f)$, we assume that division has been done, and think of the histogram as a probability function.

Figure 8.5 illustrates the same checkerboard as Figure 8.3. Two distinct peaks are present, one at gray level 100, and one at gray level 200. With the exception of noise pixels, almost every point in the image belongs to one of these regions. A good threshold, then, is anywhere between the two peaks.

Otsu's method [8.38] is very effective in analyzing histograms of bimodal images. It automatically calculates the optimal threshold that minimizes the within-class variance (or equivalently, maximizes the between-class variance. Students will have the opportunity to prove this statement in Assignment 8.2.). The procedure of Otsu's method is described below:

Otsu's Method

1. Compute the histogram and the probability, $P(i)$, of the input image, where i is the index of possible intensity levels in the input image, e.g., $i = 0, \ldots, L$.
2. For each possible threshold value, t, $t \in [0, L]$, calculate the within-class variance, $\sigma_w^2(t)$,
$$\sigma_w^2(t) = w_1(t)\sigma_1^2(t) + w_2(t)\sigma_2^2(t) \tag{8.1}$$
where $w_1(t) = \sum_{i=0}^{t-1} P(i)$ and $w_2(t) = \sum_{i=t}^{L} P(i)$ are the weights that estimate the class probabilities, and the variance of each class is calculated as
$$\sigma_1^2(t) = \sum_{i=0}^{t-1}[i - \mu_1(t)]^2 P(i)/w_1(t), \sigma_2^2(t) = \sum_{i=t}^{L}[i - \mu_2(t)]^2 P(i)/w_2(t), \tag{8.2}$$
where $\mu_1(t) = \sum_{i=0}^{t-1} iP(i)/w_1(t)$ and $\mu_2(t) = \sum_{i=t}^{L} iP(i)/w_2(t)$ are the means of each class.
3. The optimal threshold is the one that minimizes $\sigma_w^2(t)$.

Given the equivalence mentioned above, this procedure can also be revised to maximize the between-class variance,

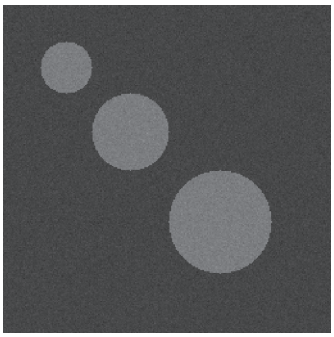
$$\sigma_b^2(t) = w_1(t)w_2(t)[\mu_1(t) - \mu_2(t)]^2. \tag{8.3}$$

The benefit of having a maximization formulation is that it allows the between-class variance to be calculated in a recursive manner, speeding up the exhaustive search procedure of Otsu's method, which is otherwise quite time consuming. The proof of Eq. 8.3 as well as its recursive calculation nature is a homework problem (Assignment 8.3). Figure 8.6 shows two synthetic images, their histograms, and the thresholding results using Otsu's method. We can see that when the histogram is bimodal, Otsu's method works very effectively even with 20dB Gaussian noise added to the image. However, if the histogram is not bimodal, e.g., exhibiting three peaks, Otsu's method fails to generate a reasonable segmentation result.

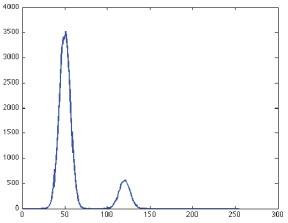
In the following sections, two more sophisticated techniques for finding the optimal threshold are described.

8.2.3 Fitting the Histogram with a Sum of Gaussians

Given a bimodal histogram, $h(f)$, of an image, where f represents a brightness value, a standard technique [8.17] to find the best threshold of that image is to fit the histogram

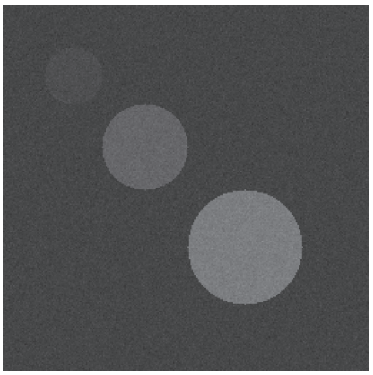


(a) Synthetic image 1.

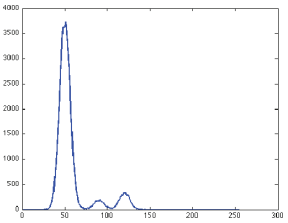


(b) Histogram.

(c) Thresholding result.



(d) Synthetic image 1.8.



(e) Histogram.

(f) Thresholding result.

Figure 8.6 Two synthetic images and the thresholding result using Otsu's method. (Library names: seg_synthetic1.png and seg_synthetic2.png)

with the sum of two Gaussians:

$$h(f) = \frac{A_1}{\sqrt{2\pi}\sigma_1} \exp\left[\frac{-(f-\mu_1)^2}{2\sigma_1^2}\right] + \frac{A_2}{\sqrt{2\pi}\sigma_2} \exp\left[\frac{-(f-\mu_2)^2}{2\sigma_2^2}\right]. \tag{8.4}$$

To fit this function to an experimentally derived histogram requires estimation of six parameters: two A's, two μ's and two σ's. If $h(f)$ is properly normalized, one may adjust the usual normalization of the two-component Gaussian so that each sums to unity on the 256 discrete gray levels (rather than integrating to unity on the continuous interval), and thereby admit the additional constraint that $A_1 + A_2 = 1$. Use of this constraint reduces the number of parameters to be estimated from 6 to 5.

Simple fitting would follow an approach such as the one presented in Chapter 5 to fit a surface to data in order to find a kernel. Unfortunately, the complexity of the sum-of-Gaussians expression makes a simple approach infeasible. Instead, we must use some sort of numerical minimization. One thinks immediately of gradient descent, but conventional descent often terminates at a suboptimal local minimum for this two-Gaussian problem and is even less reliable for a three-Gaussian problem. The results in Figure 8.7 were obtained by the authors using a nonlinear optimization method called *tree annealing* (TA) [8.7, 8.47], which deals easily with the two- or three-Gaussian problem.

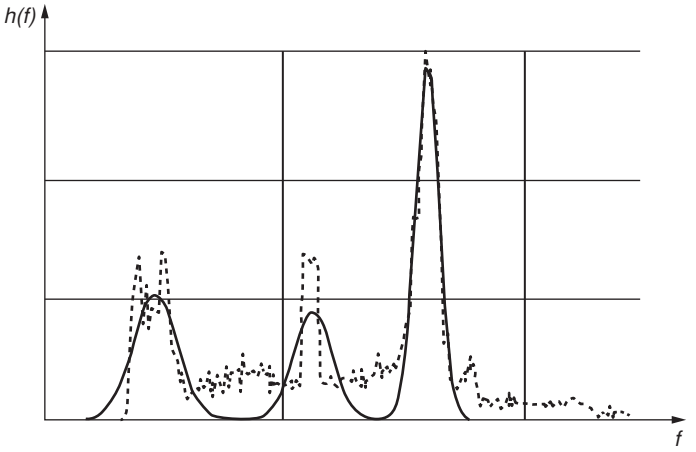


Figure 8.7 MSE fit of the sum of three Gaussians to a histogram, $h(f)$. The center peak was added artificially to demonstrate that the algorithm can fit arbitrarily complex functions.

Whatever algorithm we use, the philosophy of histogram-based thresholding is the same: find peaks in the histogram and choose the threshold to be between them.

The problem described in this section is a particular case of a more general problem called *Gaussian Mixture Models* (GMM's), that is, using a sum of Gaussians to approximate a function. This section also provides a context in which to describe a powerful and general optimization method called *Expectation Maximization* (EM). In the next subsection, we illustrate the use of EM to solve the same problem we just addressed, finding the parameters of a Gaussian Mixture Model.

8.2.4 Gaussian Mixture Models and Expectation Maximization

In the previous subsection, we described the Gaussian Mixture Model and provided one approach to solving for the parameters of this model. In this subsection, EM is illustrated by using it to estimate the parameters of a Gaussian Mixture Model.

Expectation Maximization is an iterative algorithm with two phases. In the first phase, usually called the "E-step," an objective that describes the system is set up, and posed as an expected value. In the second phase, the "M-step," the parameters that optimize that expected value are found. Then the process iterates.

The derivation for the EM approach to solving GMM problems in this section follows the form and order of [8.15], but here, only one-dimensional Gaussians are used.

Given n samples $y_1, y_2, \ldots, y_n$, from the sum of K Gaussians, we need to estimate A_k, μ_k, and σ_k for $k = 1, \ldots, K$. Define

$$\psi(y|\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) \tag{8.5}$$

and using ψ, define γ_{ij}^m to be an estimate of the probability that the ith sample belongs to the jth Gaussian at the m^{th} iteration:

$$\gamma_{ij}^m = \frac{A_j^m \psi\left(y_i|\mu_j^m, \sigma_j^m\right)}{\sum_{k=1}^K A_k^m \psi\left(y_i|\mu_k^m, \sigma_k^m\right)}, \tag{8.6}$$

where the superscript m on a parameter denotes the estimate of that parameter at the m^{th} iteration.

The denominator in the previous equation guarantees that $\sum_{j=1}^{K} \gamma_{ij}^m = 1$. With this constraint, we can be comfortable thinking of the γ's as probabilities. In the following, we will usually drop the superscript m, except on γ, since all operations are performed identically at each iteration.

As in section 6.5.1, we usually deal with logarithms of probabilities rather than the probabilities themselves. This is because we almost always deal with random variables assumed to be independent. This independence means joint probabilities can be written as products, and their logarithms as sums. Operations that are easy on sums (e.g., differentiation) can be quite messy when attempted on products.

Denote by θ the vector of parameters, including all the A's, μ's, and all the σ's. Initially, we simply choose some value for these variables,[1] and assume those values are correct. They are then updated as the algorithm runs.

Based on a single measurement, y_i, we seek the expected value of $\log p(y_i|\theta)$. Recall that the expected value of a random variable is a sum of the value of the random variable multiplied by its probability. Denote this value by $Q_i(\theta|\theta^m)$, the expected value of the parameters, given the m^{th} estimate of those parameters, so that

$$\begin{aligned} Q_i(\theta|\theta^m) &= \sum_{j=1}^{K} \gamma_{ij}^m \log A_j \psi(y_i|\mu_j, \sigma_j) \\ &= \sum_{j=1}^{K} \gamma_{ij}^m \left(\log A_j - \frac{1}{2}\log 2\pi - \log \sigma_j - \frac{(y_i - \mu_j)^2}{2\sigma_j^2} \right). \end{aligned} \tag{8.7}$$

It is not hard to show that if Eq. 8.7 holds for a single measurement y_i, then

$$Q(\theta|\theta^m) = \sum_{i=1}^{n} \sum_{j=1}^{K} \gamma_{ij}^m \left(\log A_j - \log \sigma_j - \frac{(y_i - \mu_j)^2}{2\sigma_j} \right) + C, \tag{8.8}$$

where C is a constant that does not affect the maximization and hence can be dropped. To get the expectation into a more tractable form define

$$n_j^m = \sum_{i=1}^{n} \gamma_{ij}^m. \tag{8.9}$$

Equation 8.9 allows us to separate the A's from the μ's and the σ's, writing

$$Q(\theta|\theta^m) = \sum_{j=1}^{K} n_j^m \left(\log A_j - \log \sigma_j \right) - \sum_{i=1}^{n} \sum_{j=1}^{K} \gamma_{ij}^m \frac{(y_i - \mu_j)^2}{2\sigma_j^2}. \tag{8.10}$$

This equation concludes the E-step of the EM algorithm.

Now that we have $Q(\theta|\theta^m)$ in a closed form, we find which values of the parameters maximize it. In this brief introduction, we illustrate the process by finding only the means, μ_j, leaving the estimation of the optimizing A's and σ's as exercises. First, we observe that

[1] See [8.15] for more discussion on initialization.

since the first term of Eq. 8.10 contains no reference to μ, we ignore it, and rewrite the equation for Q without the first term, just for the purpose of estimating the μ's.

$$Q = -\sum_{i=1}^{n}\sum_{j=1}^{K}\gamma_{ij}^{m}\frac{(y_i-\mu_j)^2}{2\sigma_j^2}. \tag{8.11}$$

Just to make the explanation clearer, reverse the order of the summations:

$$Q = -\sum_{j=1}^{K}\sum_{i=1}^{n}\gamma_{ij}^{m}\frac{(y_i-\mu_j)^2}{2\sigma_j^2}. \tag{8.12}$$

We will differentiate with respect to μ_k, and when we do so, all the terms in the sum over j will go to zero, except when $j = k$, eliminating that sum. Thus differentiating Eq. 8.10 with respect to μ_k:

$$\frac{\partial Q(\theta|\theta^m)}{\partial\mu_k} = \frac{\partial Q}{\partial\mu_k} = \sum_{i=1}^{n}\gamma_{ik}^{m}\frac{(y_i-\mu_k)}{\sigma_k^2}. \tag{8.13}$$

Set the derivative to zero:

$$\frac{1}{\sigma_k^2}\sum_i\gamma_{ik}^{m}y_i - \frac{1}{\sigma_k^2}\sum_i\gamma_{ik}^{m}\mu_k = 0. \tag{8.14}$$

Since it does not depend on i, factor μ_k out of the summation, eliminate the constants, and rearrange:

$$\mu_k\sum_i\gamma_{ik}^{m} = \sum_i\gamma_{ik}^{m}y_i \tag{8.15}$$

and therefore

$$\mu_k = \frac{\sum_i\gamma_{ik}^{m}y_i}{\sum_i\gamma_{ik}^{m}}. \tag{8.16}$$

But the denominator was already defined to be n_k, so

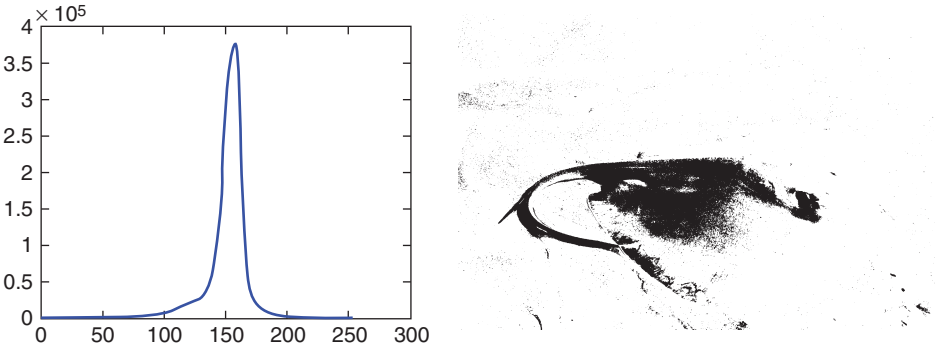
$$\mu_k = \frac{1}{n_k}\sum_i\gamma_{ik}^{m}y_i. \tag{8.17}$$

After solving for the optimizing values of the other parameters, we have completed the M-step of the EM algorithm.

These new estimates of the parameters are now used in a new calculation of the E-step, and the process iterates.

EM is guaranteed to never produce worse estimates over iterations [8.55]; however, it can become stuck in local maxima. For that reason, EM is usually started several times, from different initial conditions. See [8.21, 8.10, 8.41] for more discussion of convergence properties.

This approach will not work when the histogram cannot be adequately described by a Gaussian Mixture model. For example, Figure 8.8 shows the thresholding result using Otsu's method applied on the stingray image on page 147. For this image, simple thresholding methods completely fail, and we must resort to more complex segmentation approaches such as active contours, which will be discussed in section 8.5.



(a) Histogram. (b) Otsu's method.

Figure 8.8 A nearly unimodal histogram and thresholding results using Otsu's method.

8.3 Clustering: Segmentation Based on Color Similarity

In this section you will learn a very powerful and general tool, clustering. In section 8.2, we mainly discussed the thresholding technique applied to grayscale images. When there are multiple spectral bands (e.g., the three-band color images), we can certainly apply thresholding on each individual band and then use some fusion approach to combine segmentation results from individual bands. In this section, we describe another way to produce a segmentation of a color image, and also use this opportunity to teach clustering.

Figure 8.9 illustrates a street in the village of Appledore in North Devon, UK where several houses painted with different colors are shown. In order to segment the houses of different colors, one intuitive approach is to find regions of similar colors. That is, we can represent each house by a cluster of similar-colored pixels, and represent the cluster by a single color. One might ask: "How many colors are in this picture?" A first attempt at answering this question would be to reason that the color image contains 8 bits of red, 8 bits of green, and 8 bits of blue. Therefore, there would be $2^{24} = 16,777,216$ different colors. But a bit of meditation leads one to realize there are only $1296 \times 864 = 1,119,744$ pixels in this 1296×864 image, and therefore there really cannot be more than 1,119,744 different colors in the image.

But how many colors are really needed? Is there a way to use even fewer? These questions are addressed next.

This application gives us the opportunity to learn about *clustering*, or *unsupervised learning*. The clustering method described below readily produces the desired color partition.

8.3.1 K-means Clustering

There are many ways to do clustering. Stuart Lloyd developed what later became known as the *k-means algorithm* in 1957, but did not publish it in the open literature until 1982 [8.32, 8.33]. Interestingly, decades later, k-means is still one of the most popular clustering algorithms.

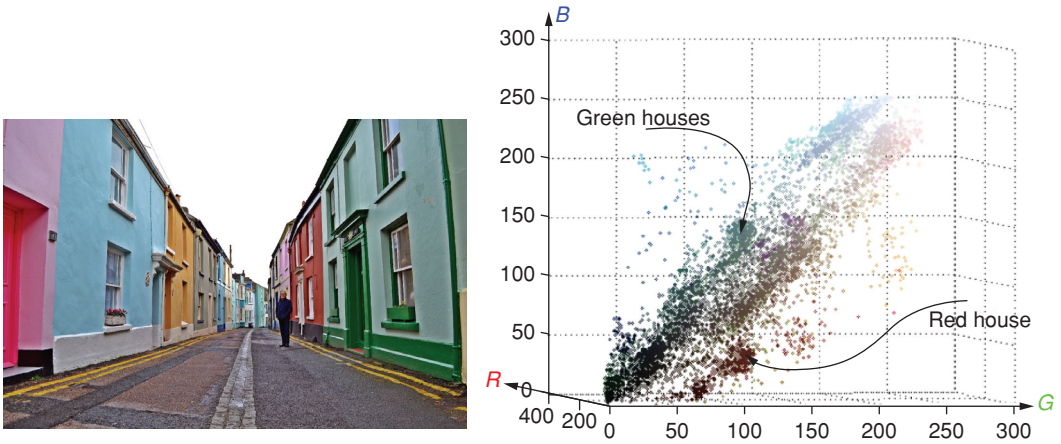


Figure 8.9 LEFT: The village of Appledore, North Devon, UK. The authors are grateful to Matthew Rowe, Irene Rowe, and David Rowe for this picture. Library Name: ColoredHouses-sm.jpg. RIGHT: The three-dimensional RGB color space for the image on the left. Each point in the color space corresponds to a pixel in the image and is shown at coordinates, $< r, g, b >$ with color $< r, g, b >$.

The student is advised (again) to take a course in pattern recognition to gain an understanding of this concept in more depth.

The k-means algorithm is presented in Figure 8.10:

Performance of color clustering has been compared to the problem Michelangelo would have had if he had been required to color the Sistine chapel with crayons. Figure 8.11 illustrates the results of using a small box of 10 crayons, a medium box of 24 crayons,

1. In the 3D color space $< r, g, b >$, choose a set of k initial cluster centers. How you pick the initial cluster centers is problem-dependent. For example, if you are clustering points in a color space, where the dimensions are red, green, and blue, you might scatter all your cluster centers uniformly over this 3-space, or you might put all of them along the line from <0,0,0> to <maxred, maxgreen, maxblue>.
2. Assign each pixel in the image as belonging to the closest cluster center. That is, for the pixel at $< x, y >$, whose color is $< r, g, b >$, compute the distance to cluster mean i whose color is r_i, g_i, b_i, using
$$d_{i,x,y} = \sqrt{(r - r_i)^2 + (g - g_i)^2 + (b - b_i)^2}. \quad (8.18)$$
The pixel at $< x, y >$ is said to "belong" to cluster i if $d_{i,x,y} < d_{j,x,y}$ for any cluster j, $j \neq i$.
3. Compute the mean of each cluster, that is, the color that is the average of all the colors in that cluster. This becomes the new center of that cluster.
4. Reassign each pixel to the cluster with the nearest mean.
5. If nothing has changed in this iteration, exit; otherwise go back to step 2.

Figure 8.10 K-means clustering of colors.

Figure 8.11 "Coloring" Figure 8.9 using 10, 24, and 64 different colors. To the human eye, the 64 color rendering is indistinguishable from the original, and the 24 color is almost indistinguishable. Exact results depend on the choice of initial color centers, because this algorithm is sensitive to local minima.

and a large box of 64 crayons. That is, we have color segmentations with 10, 24, and 64 clusters, respectively. The color of each crayon is the color of the cluster center.

Although Lloyd's algorithm is both simple and well known, there are three major issues with the algorithm design. First, it assumes the prior knowledge of the number of clusters (i.e., k) that in many cases becomes inconvenient and unrealistic. Second, the algorithm is very sensitive to the initial cluster selection. Figure 8.12 shows the result of color segmentation using random initialization and uniform distribution along the diagonal direction. The difference in the output result is apparent. Third, the algorithm has a strong bias toward finding (hyper)spherical clusters. For example, it does not do well on clustering problems such as the one shown in Figure 8.13.

For clusters like those, other algorithms are better suited, such as the *mean shift* algorithm, which will be discussed next. Another option is the set of *hierarchical methods*, which are beyond the scope of this text. We should note that there is a body of literature that the student may encounter about *Vector Quantization*, which has many similarities to clustering.

8.3.2 Mean Shift Clustering

The mean shift algorithm was originally proposed in 1975 by Fukunaga and Hostetler [8.23] for mode detection. It considers the set of input points as samples drawn from an

(a) Initialization using random number between 0 and 1.

(b) Uniform initialization along the diagonal of the 3D color plane.

Figure 8.12 Effect of different initializations to color clustering using k-means ($k = 6$). Library Name: ColoredHouses-sm.jpg.

empirical probability density function. The algorithm iteratively shifts each data point to the "mode" (or local maximum) of the probability density function. Although powerful and versatile, the algorithm had not drawn much attention until Cheng's work [8.16] in 1995, which showed that mean shift could be generalized to solve clustering problems.

Unlike the k-means algorithm, mean shift does not assume prior knowledge of the number of clusters, due to the nonparametric nature of the algorithm. The number of modes automatically indicates the number of clusters. In addition, since mean shift is based on density estimation, it can work on arbitrarily shaped clusters.

The key parameter in the mean shift algorithm is the so-called *window size*, h, that specifies the radius or size of the window/kernel function. The mean value is calculated within each window. The window size determines, indirectly, the distance between clusters. Hence instead of having to predefine the number of clusters as in the k-means algorithm, the mean shift algorithm automatically yields the clustering results based on the specified window size. The mean shift algorithm is described in Figure 8.14.

Cheng [8.16] showed that if one uses a Gaussian kernel instead of a flat one, the mean shift procedure is equivalent to a gradient ascent process, making clustering analysis a deterministic problem of finding a fixed point of mean shift that characterizes the data.

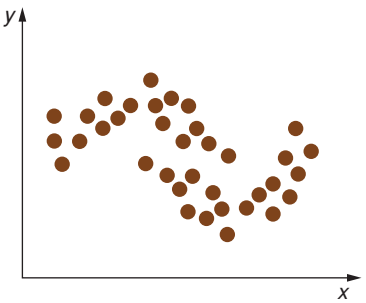


Figure 8.13 Two clusters that are elongated and twisted, and cannot be clustered accurately by k-means.

1. Initialization: Choose a window/kernel of size h, e.g., a flat kernel,
$$K(x) = \begin{cases} 1 & \text{if } ||x|| \leq h \\ 0 & \text{if } ||x|| > h \end{cases}$$
and apply the window on each data point, x.
2. Mean calculation: Within each window centered at x, compute the mean of data, $m(x)$.
$$m(x) = \frac{\sum_{s \in \Omega_x} K(s-x)s}{\sum_{s \in \Omega_x} K(s-x)}$$
where Ω_x is the set of data points included within the window specified by h.
3. Mean shift: Shift the window to the mean, i.e., $x = m(x)$, where the difference $m(x) - x$ is referred to as the *mean shift* in [8.23].
4. If $\|m(x) - x\| > \epsilon$, go back to Step 2.

Figure 8.14 Mean shift clustering of colors.

Figure 8.15 illustrates the clustering result using mean shift. The biggest issue with mean shift is its slow convergence rate and the need to tune the window size parameter, h. There are algorithms [8.19] that adaptively adjust the window size depending on the local density distribution of the data.

8.4 Connected Components: Spatial Segmentation Using Region Growing

In this section, you will learn one way to distinguish one region from another in an image. This will be related to whether the first region is "connected" to the second.

"Is this region connected to that region?" is a problem that can be addressed in many ways. Probably the simplest algorithm is known as "region growing." It utilizes a label memory, L, isomorphic to the image memory, f, just as Figure 8.2 corresponds to Figure 8.1. In this description, we will refer to "black" pixels as object and "white" pixles as background.

(a) 14 clusters with window size $h = 30$. **(b)** 35 clusters with window size $h = 20$.

Figure 8.15 Clustering result using mean shift where the number of clusters is automatically determined.

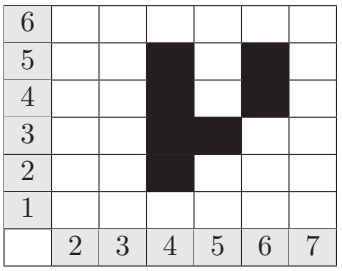


Figure 8.16 An example foreground/background image.

8.4.1 A Recursive Approach

This algorithm implements region growing by using a pushdown stack on which to temporarily keep the coordinates of pixels in the region. It produces a label image, $L(x, y)$ from an image $f(x, y)$. L is initialized to all zeros.

1. Find an unlabeled black pixel; that is, $L(x, y) = 0$. Choose a new label number for this region, call it N. If all pixels have been labeled, stop.
2. $L(x, y) \leftarrow N$.
3. If $f(x-1, y)$ is black and $L(x-1, y) = 0$, set $L(x-1, y) \leftarrow N$ and push the coordinate pair $(x-1, y)$ onto the stack.
   If $f(x+1, y)$ is black and $L(x+1, y) = 0$, set $L(x+1, y) \leftarrow N$ and push $(x+1, y)$ onto the stack.
   If $f(x, y-1)$ is black and $L(x, y-1) = 0$, set $L(x, y-1) \leftarrow N$ and push $(x, y-1)$ onto the stack.
   If $f(x, y+1)$ is black and $L(x, y+1) = 0$, set $L(x, y+1) \leftarrow N$ and push $(x, y+1)$ onto the stack.
4. Choose a new (x, y) by popping the stack.
5. If the stack is empty, increment N and go to 1, else go to 2.

This labeling operation results in a set of connected regions, each assigned a unique label number. Note that this example assumes a 4-neighbor adjacency. To find the region to which any given pixel belongs, the computer has only to interrogate the corresponding location in the L memory and read the region number.

Example: Applying Region Growing

The algorithm described here is called *Region Growing* because it identifies which pixels belong to a partial region one pixel at a time, effectively "growing" the region in the label image. In Figure 8.16 a 6×6 array of pixels is shown. Assume the initial value of $< x, y >$ is $< 4, 5 >$. We apply the above algorithm and show the contents of the stack and L each time step 3 is executed. Let the initial value of N be 1. In the figures below, row and column numbers are shaded with gray.

Here, we illustrate the concept of connectivity by defining that two pixels are CONNECTED if they are the same brightness and the Manhattan distance between them is 1.

6	0	0	0	0	0	0
5	0	0	1	0	0	0
4	0	0	0	0	0	0
3	0	0	0	0	0	0
2	0	0	0	0	0	0
1	0	0	0	0	0	0
	2	3	4	5	6	7

(a) Pass 1 with stack: $< 4, 4 >$

6	0	0	0	0	0	0
5	0	0	1	0	0	0
4	0	0	1	0	0	0
3	0	0	0	0	0	0
2	0	0	0	0	0	0
1	0	0	0	0	0	0
	2	3	4	5	6	7

(b) Pass 2 with stack: $< 4, 3 >$

6	0	0	0	0	0	0
5	0	0	1	0	0	0
4	0	0	1	0	0	0
3	0	0	1	0	0	0
2	0	0	0	0	0	0
1	0	0	0	0	0	0
	2	3	4	5	6	7

(c) Pass 3 with stack: $< 5, 3 >$ on top of $< 4, 2 >$

6	0	0	0	0	0	0
5	0	0	1	0	0	0
4	0	0	1	0	0	0
3	0	0	1	1	0	0
2	0	0	0	0	0	0
1	0	0	0	0	0	0
	2	3	4	5	6	7

(d) Pass 4 with stack: $< 4, 2 >$

6	0	0	0	0	0	0
5	0	0	1	0	0	0
4	0	0	1	0	0	0
3	0	0	1	1	0	0
2	0	0	1	0	0	0
1	0	0	0	0	0	0
	2	3	4	5	6	7

(e) Pass 5 with stack: $<>$

Figure 8.17 Content of the L memory after each pass in the example.

Solution: See Figure 8.17 to follow along with the algorithm.

Pass 1: Immediately after execution of step 3. N has been set to 1. The algorithm has examined pixel $< 4, 5 >$, examined its 4-neighbors, and detected only one 4-neighbor in the foreground, the pixel at $< 4, 4 >$. Thus, the coordinates of that pixel are placed on the stack.

Pass 2: The $< 4, 4 >$ is removed from the top of the stack and marked with a 1 in the L image, its neighbors are examined and only one 4-neighbor is found, the coordinates $< 4, 3 >$ are pushed onto the stack.

Pass 3: The top of the stack contains $< 4, 3 >$. That coordinate pair is removed from the stack and marked with a 1 in the L image. All its neighbors are examined and two 4-neighbors are found, the pixels at $< 4, 2 >$ and $< 5, 3 >$. Thus the coordinates of these pixels are put on the stack.

Pass 4: The stack is "popped" again, this time removing the $< 5, 3 >$ and marking the corresponding point in the L image with a 1. That pixel is examined and determined to have no 4-neighbors that have not already been labeled.

Pass 5: The stack was popped yet again, removing the $< 4, 2 >$ and marking that point in the L image with a 1. That pixel is examined and determined to have no 4-neighbors that have not already been labeled.

Pass 6: The stack is popped again, this time producing a return value of "stack empty" and the algorithm is complete for this region since all black pixels connected to the original pixel $< 4, 5 >$ have been labeled. Note that the two pixels at $< 6, 4 >$ and $< 6, 5 >$ have not been labeled.

Now, N is incremented to 2 so that the next region will be labeled with a 2. Then, scanning is resumed. Eventually, the two unlabeled black pixels will be encountered and labeled by the same stack-based algorithm.

The simplest generalization of the algorithm above is to allow a more general definition of "connected to." That is, in the description above, foreground pixels were all black. If we allow grayscale pixels, we could simply define "connected to" with a predicate like

$$|f(\mathbf{x}) - f(\mathbf{y})| < R \Rightarrow CONNECTED(\mathbf{x}, \mathbf{y}), \tag{8.19}$$

or, a more complex measure could be used, comparing, for example, pixel color. This region-growing algorithm is just one of several strategies for performing *connected component analysis*. Other strategies exist that are faster than the one described, including one that runs at raster scan rates [8.49].

8.4.2 An Iterative Approach

Because the region-growing technique described above always results in closed regions, it is usually preferable to other segmentation techniques that are based on edge detection or line fitting. Numerous variations and applications of the basic region-growing technique have been proposed [8.11, 8.22]. Although region growing has proved to be an integral part of Computer Vision, it is not well suited for real-time application. This has prompted the consideration of alternative, faster, more hardware-specific methods of region partitioning.

This section presents an alternative algorithm that also produces closed regions. This algorithm is functionally identical to recursive region growing in that it returns a set of labeled pixels, meeting the adjacency and similarity criteria. This method was first published [8.49] as a graph manipulation method based on the graph theoretic concepts of "union-find." It is attractive because it runs at video rates when implemented by special-purpose hardware [8.48]. In addition, the algorithm will produce the label image "on the fly" with a single pass over the data. This result is achieved by using the concept of a content-addressable memory. This memory may be a physical piece of hardware or a lookup-table-driven access method in simulation software.

This algorithm is based on the concept of equivalence relationships between the pixels of an image. Equivalency is defined here as follows: Two pixels a and b are defined to be equivalent (denoted $R(a, b)$) if they belong to the same region of an image. This relationship can be shown to be reflexive ($R(a, a)$), symmetric ($R(a, b) \Rightarrow R(b, a)$), and transitive ($R(a, b) \bigwedge R(b, c) \Rightarrow R(a, c)$); which makes it an *equivalence relation*. The transitive property of an equivalence relation enables all pixels in a region to be determined by considering only local adjacency properties. In this algorithm, the fact that belonging to the same region is transitive allows inference.

As in the previous section, a label image named L will hold the label for a region. The label memory is initialized to 0, as is a counter named N. In addition, an array M is created and is interpreted as meaning: *if* $M[i] = k$, *then label* i *is equivalent to label* k. Following this interpretation, if M is initialized by $\forall i, M[i] = i$, implying that no two labels are equivalent initially. Pixels are labeled top-to-bottom, left-to-right, in a raster scan order. A foreground pixel i is compared to the pixel above and the pixel to its left. Three scenarios can occur:

1. If neither pixel is labeled, which means they are background pixels, and the current pixel is the start of a new region, increase N by 1, and assign N as the label for pixel i.

```
1  1  1        2  2
1  1  1        2  2
1  1  1        2  2
1  1  1        2  2
1  1  1  1  1  ?
```

Figure 8.18 Ambiguity in label assignment.

2. If the two pixels have similar brightnesses, they will be considered "equivalent." If the pixel above and the pixel to the left are already labeled, with identical labels, then label pixel i with that label also.
3. If both the pixel above and the pixel to the left are already labeled and the labels are different, it will be necessary to resolve this. Figure 8.18 demonstrates the situation that can arise as result of this comparison when the equivalence relation $R(1, 2)$ is discovered at the pixel designated by the question mark.

Labeling conflicts are resolved in the following way. Suppose we discover that a new pixel, could be labeled either r or s. Without loss of generality, we assume $r < s$. Then for every element of M with label r, replace its label with s. In this way, the equivalence label for every equivalence will be the same, which really says that all the pixels labeled with r or s are in the same region.

This algorithm above is potentially very fast, since the relabeling in Step 3 can be accomplished in a single step, using a content-addressable memory [8.48].

8.4.3 An Example Application

Figure 8.19 illustrates a range image from the Microsoft Kinect® camera, demonstrating artifacts from the camera. These artifacts are small regions with brightness of zero. Running connected component labels on the image produces a label image in which each region

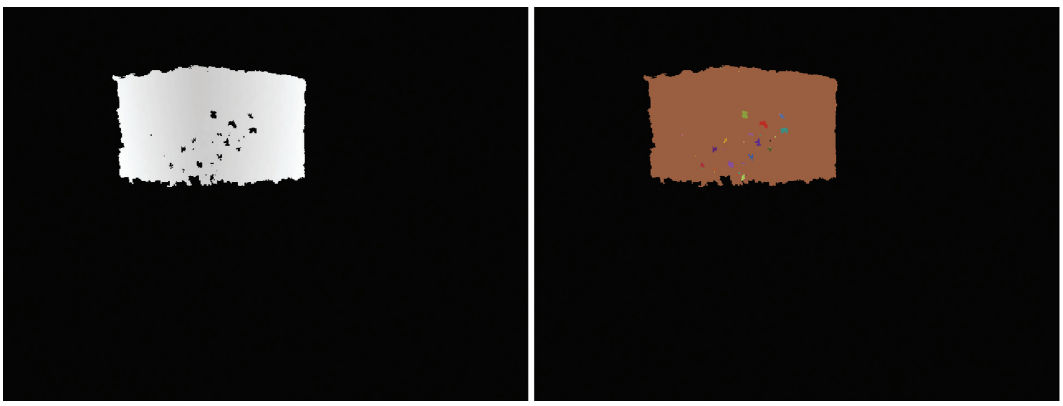


Figure 8.19 LEFT: A range image with artifacts. Library name: d_box1.png. RIGHT: The label image after assigning a different label to each artifact and then assigning random pseudocolors to the labels.

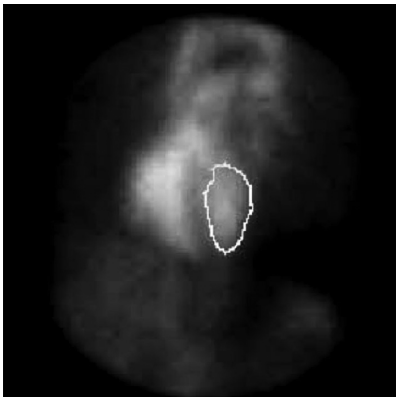


Figure 8.20 The left ventricle imaged using nuclear medicine. From [8.20], used with permission.

has a different label. Assigning a different random color to each label produces the second image in the figure.

Since all regions are now uniquely identified, they may be separately processed. For example, one might need to erase small regions completely inside larger regions. Many other similar operations are based on connected component labeling.

8.5 Segmentation Using Active Contours

In this section, you will learn how to segment regions by first assuming a contour, e.g., a circle outside the region, and then letting that contour move until it stops when it encounters the region. The concept of active contours was originally developed to address the fact that any edge detection algorithm will fail on some images, because in certain areas of the image, the edge simply does not exist. For example, Figure 8.20 illustrates a human heart imaged using nuclear medicine. In certain cardiology studies it is necessary to estimate the volume of blood in the left ventricle. The volume measurement process is performed as follows:

A radioactive drug is introduced into the blood stream, and an image is made that reflects the radiation at each point. The brightness at a point is a measure of the integral in a direction perpendicular to the imaging plane of the amount of blood in the area subtended by that pixel. The volume of blood within the ventricle can thus be calculated by summing the brightness over the area of the ventricle. Of course, this requires an accurate segmentation of the ventricle boundary, a problem made difficult by the fact that there is essentially no contrast in the upper left corner of the ventricle. This occurs because radiation from other sources behind the ventricle (superior and inferior vena cava, etc.) contributes to blur out the contrast. Thus, a technique is required that can bridge these rather large gaps.

Following this philosophy, an artificial contour is first initialized, either by a user or automatically. The contour then moves, and continues to move until many/most of the contour points align with image edge points.

Two different philosophies are discussed below that derive active contour algorithms: particle-based methods like snakes, and level sets. In particle-based methods, a set of

ordered points is first placed around the region of interest. The set of points then evolves to a new position based on energy minimization. The final contour of the object can be obtained by interpolation. An animation of the contour as it searches for boundary points reminds the viewer of the movements of a snake, hence these boundaries are often referred to as "snakes." In level set methods, on the other hand, a function that represents a smooth surface is first determined. This function is then evolved according to some partial differential equation. The final contour of the object is given by the zero level set of that function.

8.5.1 Snakes: Discrete and Continuous

A snake is an ordered set of points in the plane, $\mathbf{x}_i = [x_i \quad y_i]^{\mathrm{T}}$, which constitutes the boundary curve around some area of interest. Following this approach, the boundary moves to reduce an energy:

$$E = E_I + E_E, \tag{8.20}$$

where the internal energy (E_I) characterizes the snake itself, and the external energy (E_E) characterizes the image at the points where the snake is currently located. We will illustrate two approaches to the use of snakes to segment images: discrete snakes and continuous snakes. The former is straightforward and easy to grasp. The latter is slightly more complex, and also is somewhat more general.

The Discrete Snake

The internal energy, E_I, measures the degree of bending along the contour, the size of the curve, etc., but in particular, the internal energy does not depend on the image. The exact form of the internal energy is application-dependent, but the following form was originally used:

$$E_I = \sum_i \alpha \left\| \mathbf{x}_i - \mathbf{x}_{i-1} \right\| + \beta \left\| \mathbf{x}_{i-1} - 2\mathbf{x}_i + \mathbf{x}_{i+1} \right\|. \tag{8.21}$$

Minimizing the first term, that is, finding a set of snake points that makes $\sum \alpha \left\| \mathbf{x}_i - \mathbf{x}_{i-1} \right\|$ as small as possible, produces a curve where the snake points are close together. If $\beta = 0$, the snake will collapse to a single point (if there is no external energy).

To understand the second term, look at Figure 5.5, which is the second derivative of a Gaussian. Note that it is strongly negative in the middle. Now compare that to the coefficients inside the second term of Eq. 8.21. You can see that the second term in the internal energy roughly estimates the second derivative. Therefore, it will be minimized by a straight line, or at least one that curves slowly. Thus the curve that minimizes the second term will have little bending.

We can choose many forms for external energy. The basic idea is, we don't want the snake to move if the local image gradient is high. This could be implemented by choosing a form as simple as

$$E_E = -\sum_i \exp\left[(f_x(\mathbf{x}_i))^2 + (f_y(\mathbf{x}_i))^2 \right], \tag{8.22}$$

or

$$E_E = \sum_i \exp\left[-(f_x(\mathbf{x}_i))^2 - (f_y(\mathbf{x}_i))^2\right]. \tag{8.23}$$

Since the second form has a range limited to between zero and one, it is a bit easier to manage.

Again, lots of options are possible for a way to find a snake that minimizes E. The simplest one is *simple descent*. That is,

Finding the Minimizing Snake by Simple Descent

1. At iteration k, evaluate E for the current snake, denote that as E^k.
2. Choose a snake point at random. The use of a random choice is important.
3. Move that snake point one pixel up, down, left, or right with the choice randomized.
4. Evaluate E at the new point, call it E^{k+1}.
5. If $E^{k+1} - E^k < 0$, this is a good move, and the energy went down. So *accept* this point. That is, replace the old value of $\mathbf{x}^k$ with the new one.
6. If something has changed, go to step 1.

Next, we examine the idea of converting the discrete formulation to a continuous one. This will give us the opportunity to do the math more easily.

Discrete to Continuous Representations

In Computer Vision, we find it more convenient in some places to use a discrete problem representation with summations, and in other places to use a continuous representation with integrals. In general, it is easier to do advanced math in the continuous form, and the continuous representation often allows better insight into the function. However, algorithms must eventually be implemented in discrete form.

In this section, we present this very general concept through an example illustrating the snakes of the previous section. The student is given the opportunity to follow a derivation that shows equivalences of the two representations, and the student can learn how to switch back and forth when needed.

As we have seen before, we now have a function whose minimum is the desired contour. We need only find the set of points $\{\mathbf{x}_i\}$ that will minimize the E of Eq. 8.20.

Earlier, the observation was made that the second term of Eq. 8.21 resembled a second derivative. Now look at the first term – what a surprise! It looks like a first derivative, which leads one to consider the advisability of representing such a function in the continuous domain, as

$$E_I = \alpha \int_0^1 |\mathbf{x}'|^2 ds + \beta \int_0^1 |\mathbf{x}''|^2 ds \tag{8.24}$$

where $\mathbf{x} = \mathbf{x}(s) = [x(s)\, y(s)]^{\mathrm{T}}$, $\mathbf{x}' = \frac{\partial \mathbf{x}}{\partial s}$, $\mathbf{x}'' = \frac{\partial^2 \mathbf{x}}{\partial s^2}$, and s is the parameter of the curve. Arc length from some arbitrary starting point is normally used as the parameter. A curve is defined that passes through each snake point.

Note also that since $\mathbf{x}$ is a 2-vector, the squared norm in Eq. 8.24 is simply $|\mathbf{x}|^2 = \mathbf{x}^T\mathbf{x}$. In order to move the snake in the direction of minimizing E using gradient descent, a gradient is needed. A bit of calculus [8.26] produces, for every snake point,

$$\frac{\partial \mathbf{x}}{\partial t} = \alpha \frac{\partial^2 \mathbf{x}}{\partial s^2} - \beta \frac{\partial^4 \mathbf{x}}{\partial s^4} \tag{8.25}$$

where the derivatives may be estimated using

$$\begin{aligned} \left.\frac{\partial^2 u}{\partial s^2}\right|_j &\approx \frac{u_{j+1} - 2u_j + u_{j-1}}{\delta s^2} \\ \left.\frac{\partial^4 u}{\partial s^4}\right|_j &\approx \frac{u_{j+2} - 4u_{j+1} + 6u_j - 4u_{j-1} + u_{j-2}}{\delta s^4}. \end{aligned} \tag{8.26}$$

In Eq. 8.26, the left-hand side denotes the derivative estimated at the jth snake point. u could be either x or y. Both are needed. The estimates derived in [8.26] are estimated using *finite differences*, which is the approximation style of Eq. 5.7, provided here again as Eq. 8.27 for convenience with changed symbols to be consistent with the ones used in Eq. 8.26. Since finite differences appear so often, we will take this opportunity to briefly talk about them.

We learned the general form for the first derivative in Chapter 3:

$$\frac{\partial u}{\partial s} = \lim_{\Delta s \to 0} \frac{u(s + \Delta s) - u(s)}{\Delta s}. \tag{8.27}$$

The asymmetric approximation to the first derivative, without doing any averaging is,

$$\left.\frac{\partial u}{\partial s}\right|_j \approx \frac{u_{j+1} - u_j}{\delta s}.$$

In Chapter 5, we also saw the symetric form,

$$\left.\frac{\partial u}{\partial s}\right|_j \approx \frac{u_{j+1} - u_{j-1}}{2\delta s}.$$

To get the second derivative, we could simply take the derivative of the first derivative (using the asymmetric form):

$$\left.\frac{\partial^2 u}{\partial s^2}\right|_j \approx \frac{u_{j+1} + u_{j-1} - 2u_j}{\delta s^2}. \tag{8.28}$$

In noisy images the use of finite differences in this way is not as effective a technique for estimating derivatives as the DoG of section 5.8.1, but here, it is quite sufficient.

We need to think a bit about δ's. When we initialize the snake, if we initialize to a circle, all the snake points are equally spaced. Denote by δs_{ij} the distance between snake point i and j, and after a circular initialization, $\forall_i$, $\delta s_{i,i+1} = \delta_s$. However, after the snake begins to deform to match the image, all the δs's are no longer necessarily identical, and the set of Eq. 8.26 may not be precisely correct. In this case, you have three choices: (1) assume they are the same and move on; that may not be such a bad estimate; (2) resample the snake; or (3) calculate the derivatives using the actual distances between snake points in the following way: By first defining $s_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||$, and using the symmetric form for the

derivative, the finite difference estimate of the first two derivatives becomes

$$u'\big|_i = \left.\frac{\partial u}{\partial s}\right|_i \approx \frac{u_{i+1} - u_{i-1}}{s_{i+1} - s_{i-1}}. \tag{8.29}$$

and

$$u''_i \approx \left.\frac{\partial^2 u}{\partial s^2}\right|_i \approx \frac{(u_{i+2} - u_i)(s_i - s_{i-2}) - (u_i - u_{i-2})(s_{i+2} - s_i)}{(s_{i+2} - s_i)(s_i - s_{i-2})(s_{i+1} - s_{i-1})}, \tag{8.30}$$

which is a bit messier but more accurate than either just assuming it is close enough to uniform, or resampling the snake. Interestingly, above the first derivative, the coefficients are the same for both the asymmetric and symmetric forms.

The external energy measures properties of the image that should affect the movement of the snake. One obvious image property is the *edginess* of the region through which the boundary passes. If a region in an image is said to have high edginess, then that region contains one or more strong edges.

Equation 8.25 may be modified to include an image component as:

$$\frac{\partial \mathbf{x}}{\partial t} = \alpha \frac{\partial^2 \mathbf{x}}{\partial s^2} - \beta \frac{\partial^4 \mathbf{x}}{\partial s^4} + \mathbf{f} \tag{8.31}$$

where $\mathbf{f}$ is a force pointing toward image edges. Remember that $\delta\mathbf{x}$ is a step in the direction of an image feature of interest – here we consider only image edges – and if we add to it a vector pointing toward the edge, we modify $\delta\mathbf{x}$ in the appropriate way. The image gradient itself is one such vector, but the gradient has a large magnitude only in the vicinity of the edge, so the gradient is not effective to pull on snakes from far away. If a snake is set up around a shape and allowed to shrink using only internal forces, this may work well.

Since the gradient of the Distance Transform points toward[2] the closest edge, this may be usable in this context.

Finally, remember to update the position of each snake point using

$$\mathbf{x}^{new} = \mathbf{x}^{old} + \frac{\partial \mathbf{x}^{old}}{\partial t}. \tag{8.32}$$

In Figure 8.21, three steps in the evolution of a snake are illustrated.

There are two major problems with this type of active contour: (1) The initial contour must be close to the image edge (unless some operator is available to extend the affect of edges such as the distance transform) and (2) the contour will not progress into concave regions; in fact, the resulting contour may be the convex hull. This is illustrated in Figure 8.21.

8.5.2 Level Sets: With and without Edges

Considering only the movement of boundary points, as snakes do, introduces some additional problems. In particular, there is no really effective way to allow for the possibility that the boundary might divide into separate components, or that separate components might merge into one boundary. During the process of evolution, the boundary points can

[2] Or away from, depending on definition.

Figure 8.21 Three steps in the evolution of a snake. On the left, the initial snake, only 32 points here, is illustrated as a circle about the region of interest. In the center, the snake is shown midway through the process. It has stopped moving where it encountered the boundary on the lower left and right. On the right, the final state of the snake demonstrates one of the problems with a simple snake like this: it cannot effectively close concave regions. (Library name of original: blob.png)

become very close or very far apart, making the snake calculation numerically unstable. That is, methods based on individually moving points or *particles* cannot effectively handle topology changes of the contour.

In this section, the concept of level sets that resolve these issues is described. We present two different segmentation algorithms, both of which use the idea of level sets. First, we explain how this concept can make use of image gradients to control the level set movement. This was initially developed by Osher and Sethian [8.37]. Then, we describe a second approach that does not use image edges, but rather uses the difference in region statistics. This was initially developed by Chan and Vese [8.13].

As with snakes, we create an artificial *contour* about the area of interest, and then let that contour evolve as a function of the image. The difference is that instead of relying on the evolution of a set of particles to interpolate the contour, here, we deal with an implicit function ψ that describes a surface whose zero level set is the contour.

Given a function $\psi : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, on the x, y plane, the C *level set* of ψ is defined to be the set of points x, y, where $\psi(x, y) = C$. Figure 8.22 illustrates a level set function (Eq. 8.33) that defines a 3D surface and its three level sets with C set as 0.2, 0, and -0.5, respectively. We observe from Figure 8.22(b) to (d) that the contour first splits into two that then eventually merge into one. Topological changes of boundary contours like these

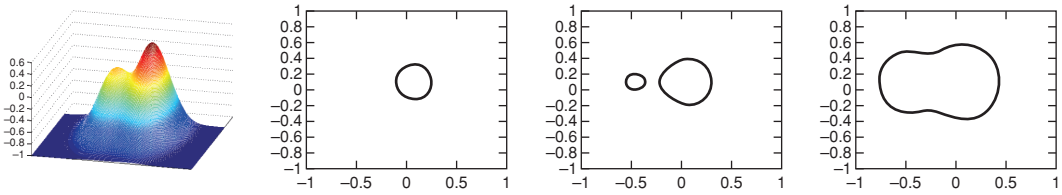


Figure 8.22 Illustration of a level set function. From (b) to (d), the contour splits into two closed curves that then merge again.

can be easily handled by the level set function, but cannot be handled by particle-based methods.

$$\psi(x,y) = 1.5e^{-5(2(x+0.5)^2+(y-0.1)^2)} + e^{-5(2(x-0.1)^2+(y-0.1)^2)} - 1 \tag{8.33}$$

The idea is this: define a function whose zero level set is the initial contour, which we get to choose. The zero level set naturally results in a segmentation of the image with two regions defined as $\{\psi(x,y) \geq 0\}$ and $\{\psi(x,y) < 0\}$. Then modify that function in such a way that its zero level set moves in the desired direction.

In practice, there are many functions of x and y that can be used. For example, we can interpret the thresholding-based segmentation methods in the context of level sets, where the level set function can be defined as the value of the grayscale image, i.e., $\psi(x,y) = f(x,y) - t$ with t being the threshold. Hence, the segmentation result is such that $\{\psi(x,y) \geq 0\}$ (or $f(x,y) \geq t$) is thought of as one region and $\{\psi(x,y) < 0\}$ (or $f(x,y) < t$) as another region. Another convenient way to define the initial function is to use the distance transform as illustrated here.

Do you remember the distance transform? In Chapter 7, the distance transform resulted in a function $DT(x,y)$ that was equal to zero on boundary pixels and got larger as one moved away from the boundary. Now consider a new version of the distance transform, which is exactly the same as the old one outside the contour of interest. (Remember, the contour is closed, and so the concepts of INSIDE and OUTSIDE make sense.) Inside the contour, this new function (which we will refer to as the metric function) is the negative of the distance transform.

$$\psi(x,y) = \begin{cases} DT(x,y), & \text{if } (x,y) \text{ is outside the contour} \\ -DT(x,y), & \text{if } (x,y) \text{ is inside the contour} \end{cases}. \tag{8.34}$$

Level set-based segmentation algorithms modify the *metric function* ψ. For every point (x,y) we compute a new value of $\psi(x,y)$, according to some modification rule. There are several ways we could modify those points, and we will discuss two of them below. One moves the contour using image edges, and the other moves the contour without using image edges. Remember that the contour of interest is still the set of points where the (modified) metric function takes on a value of zero, i.e., the zero level set. We initialize it to be the distance transform or other function of x and y. This is another advantage of the level set-based segmentation method where the initial contour does not have to be very close to the actual edge of the region of interest.

In this section, several things are related: the metric function, ψ, its zero level set, which is the contour, the motion of the contour, and the change in ψ required to result in a motion of contour. The expressions *zero level set* and *contour* are used interchangeably here. We will use the vector representation of spatial coordinates, $\mathbf{x} = [x\ y]^{\mathrm{T}}$.

Moving the Contour using Image Edges

Consider a point $\mathbf{x}$ in the zero level set, and observe that at any time t, ψ is a function of three variables,

$$\psi(x,y,t). \tag{8.35}$$

Henceforth, we will drop the argument list to make the derivation clearer. Take the complete derivative to ψ with respect to time.

$$\frac{d\psi}{dt} = \frac{\partial\psi}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial\psi}{\partial y}\frac{\partial y}{\partial t} \tag{8.36}$$

$$\frac{d\psi}{dt} = -\begin{bmatrix}\frac{\partial\psi}{\partial x} & \frac{\partial\psi}{\partial y}\end{bmatrix}\begin{bmatrix}\frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial t}\end{bmatrix}. \tag{8.37}$$

From here on, we remember that since this is all a function of time, we can use subscripts for derivatives. By writing this in terms of the change in ψ and recognizing the gradient Eq. 8.37 becomes

$$\frac{d\psi}{dt} = -(\nabla_{\mathbf{x}}\psi)^{\mathrm{T}}\mathbf{x}_t. \tag{8.38}$$

The term $\mathbf{x}_t$ describes the change in the boundary points with respect to time. Since we are free to choose this, we choose the motion to be in the direction normal to the contour. The normal vector is determined from the gradient[3]

$$\nabla_{\mathbf{x}}\psi = \left[\frac{\partial\psi}{\partial x}, \frac{\partial\psi}{\partial y}\right]^{\mathrm{T}}. \tag{8.39}$$

As you think about a level set as an isophote, you realize that the gradient is normal to the isophote, so, given the gradient vector, the normal is just the normalized (naturally) version of the gradient.[4]

$$\mathbf{x}_t = \mathbf{n}_{\mathbf{x}} = \frac{\nabla_{\mathbf{x}}\psi}{|\nabla_{\mathbf{x}}\psi|}. \tag{8.40}$$

Thus, we conclude that the change in the metric function with respect to time at a particular point, (x, y), can be written as

$$\frac{d\psi}{dt} = -S(\mathbf{x})(\nabla_{\mathbf{x}}\psi)^{\mathrm{T}}\frac{\nabla_{\mathbf{x}}\psi}{|\nabla_{\mathbf{x}}\psi|} = -S(\mathbf{x})\frac{(\nabla_{\mathbf{x}}\psi)^{\mathrm{T}}(\nabla_{\mathbf{x}}\psi)}{|\nabla_{\mathbf{x}}\psi|} = -S(\mathbf{x})\,|\nabla_{\mathbf{x}}\psi| \tag{8.41}$$

where S is a "speed" function of $\mathbf{x}$ that we have yet to define. This function will consider the smoothness of the boundary as well as the image it should match.

There are three situations for which we should design the speed function differently. First, we want the change in ψ at point $\mathbf{x}$ to approach *zero* when the image has a lot of edginess. For this to happen, define a speed as a function of the image at point $\mathbf{x}$ as something like

$$S_E = \exp\left(-||\nabla_{\mathbf{x}} f)||\right) \tag{8.42}$$

We will call this speed the *external speed* since it results from factors external to the metric function. Second, the change in ψ at point $\mathbf{x}$ should be *a small number* when the metric function is smooth. Third, the change should be *a larger number* when it has significant

[3] Be sure you understand: here, we are talking about the gradient of the metric function, not the gradient of the image.

[4] Don't get confused in the next equation. $\nabla\psi$ is an operator applied to a scalar function. It is a single thing. Think of it as if it were one symbol.

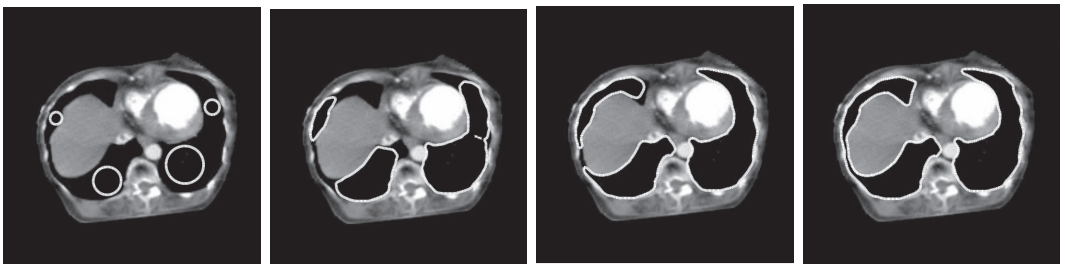


Figure 8.23 The first, on the left, is the initial, showing circular contours. In the second, through the fourth, the level set continues to grow. The authors are grateful to T. F. Chen for this figure. Used with permission. From [8.14].

variation and we want the contour to be smooth. This can be accomplished by using curvature, which tells us how rapidly the contour is changing. The curvature of the level set may be locally determined from

$$\kappa = \frac{\psi_{xx}\psi_x^2 + 2\psi_x\psi_y\psi_{xy} + \psi_{yy}\psi_y^2}{(\psi_x^2 + \psi_y^2)^{3/2}}. \tag{8.43}$$

If the denominator in this equation is zero, problems occur, and a zero denominator is not impossible, especially for synthetic images. To avoid that inconvenience, Eq. 8.43 may be stabilized by adding a small constant to the denominator, a number like 0.01. Remembering that curvature normally only takes on values between zero and one,[5] the behavior we seek can be obtained by using curvature in a way similar to this [8.44]:

$$S_I = 1 - \epsilon\kappa, \tag{8.44}$$

where ϵ is a constant used to determine the influence of the curvature, κ.

There are many ways to define speed that will function in this role. The sign of the speed may be chosen to encourage the contour to move in or out. If we define $S = S_I S_E$, Eq. 8.41 becomes

$$\frac{d\psi}{dt} = -S_I S_E |\nabla_{\mathbf{x}} \psi|. \tag{8.45}$$

Eq. 8.45 is used to update ψ by relating speed to curvature using

$$\psi^{n+1}(x, y) = \psi^n(x, y) - \alpha S(x, y) \left|\nabla_{\mathbf{x}} \psi(x, y)\right|, \tag{8.46}$$

where α is some small scalar. α is used just as in Eq. 3.40 to ensure the algorithm does not step completely over the optimal solution in a single step.

Over the course of running the algorithm, the metric function evolves following a rule like Eq. 8.46. As it evolves, it will have zero values at different points, and those points will define the evolution of the contour, i.e., the zero level set.

In Figure 8.23, four instances in the evolution of a level-set based segmentation are illustrated. The original image is a computed tomographic image of the chest. Dark areas in the chest are being identified.

[5] The curvature is 1/(radius of locally best fitting circle). A straight line has infinite radius – zero curvature, and we cannot define a circle smaller than a single pixel.

Moving the Contour without Using Image Edges

In this subsection you will learn the concepts underlying the Chan-Vese algorithm, one of the most effective current methods for active contour-based segmentation. You will also get more exposure to the process of converting between continuous and discrete mathematics.

While most algorithms have names that describe what they do, the algorithm described here is named for the authors, Chan and Vese, probably because "Segmentation by maximizing the difference between two regions" is too hard to say. At the time of this writing, extensions of the Chan-Vese algorithm provide some of the very best segmentations, competing with some graph-based methods that will be discussed later in this chapter. The Chan-Vese algorithm uses level set approaches to find the contour that divides the image into two regions that are maximally different. It is important to realize that these regions are not required to be connected. Since the algorithm does not rely on image gradient, it can successfully detect objects whose boundaries are not necessarily defined by edges.

Here, we present the same example application that was in the original paper by Chan and Vese [8.13]. There are many papers available to the student that extend the Chan-Vese algorithm [8.12, 8.18, 8.1, 8.2], just to mention a few.

Philosophically, Chan-Vese segmentation chooses an objective function that describes the difference in brightness between the outside and inside of the contour. The version discussed here assumes the algorithm is finding a contour that separates two relatively homogeneous regions, denoted *background* and *foreground*. As we did in section 8.5.1 the derivation begins using continuous mathematics and concludes using finite differences. Consider the following function:

$$F(C) = \int_{inside\ C} (f(x,y) - \mu_1)^2 dxdy + \int_{outside\ C} (f(x,y) - \mu_2)^2 dxdy, \tag{8.47}$$

where μ_1 is the mean brightness inside the contour and μ_2 is the mean brightness outside the contour.

$F(C)$ will be small if the two regions are close to homogeneous, one precisely enclosed by the contour, and the other totally outside the contour. Finding the best such contour C is the objective of the optimization.

One problem with this objective function can be identified before proceeding further: The contour that optimizes this function is not constrained to be smooth, so it may sharply curve and deform to match the two regions precisely. We modify $F(C)$ by adding a regularizing term to encourage a smooth contour – simply minimizing the length of the contour.

$$F(C) = \alpha\, Length(C) + \int_{inside\ C} (f(x,y) - \mu_1)^2 dxdy + \int_{outside\ C} (f(x,y) - \mu_2)^2 dxdy, \tag{8.48}$$

where α is a constant scaling the relative importance of the length of the contour compared to the difference in region statistics.

This problem can be readily converted to a level set problem by defining the function $\psi(x,y)$ as we did previously. For convenience, we denote the set of points inside the contour

as ω, and all the points in the image as Ω.

$$\begin{aligned} \omega &= \{(x,y)|\psi(x,y) > 0\} \\ \omega^c &= \{(x,y)|\psi(x,y) < 0\}. \\ C = \partial\omega &= \{(x,y)|\psi(x,y) = 0\} \end{aligned} \tag{8.49}$$

(The student should observe that this is a slightly different definition of ψ from that given at the beginning of section 8.5.2 because here, the inside is positive and the outside negative.)

To convert Eq. 8.48 into something that can be minimized, we introduce a binary function,

$$H(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0. \end{cases} \tag{8.50}$$

$H(.)$ is called the *Heaviside function*, and has a derivative by definition:

$$\delta(z) = \frac{\partial H(z)}{\partial z}, \tag{8.51}$$

where δ is the Dirac measure.

Now we can write the previous expressions in terms of H, remembering that the contour of interest is the zero level set of ψ:

$$\begin{aligned} Length(C) = Length(\psi = 0) &= \int_\Omega |\nabla H(\psi(x,y))| dxdy \\ &= \int_\Omega \delta(\psi(x,y))|\nabla\psi(x,y)| dxdy. \end{aligned} \tag{8.52}$$

We get rid of the nonuniform regions of integration by using H:

$$\int_{\psi>0} (f(x,y) - \mu_1)^2 dxdy \text{ becomes } \int_\Omega (f(x,y) - \mu_1)^2 H(\psi(x,y)) dxdy \tag{8.53}$$

and similarly

$$\int_{\psi<0} (f(x,y) - \mu_2)^2 dxdy \text{ becomes } \int_\Omega (f(x,y) - \mu_2)^2 (1 - H(\psi(x,y))) dxdy. \tag{8.54}$$

Finally, collecting the components of F, we have

$$\begin{aligned} F(C) = \alpha &\int_\Omega \delta(\psi(x,y))|\nabla\psi(x,y)| dxdy \\ &+ \int_\Omega (f(x,y) - \mu_1)^2 H(\psi(x,y)) dxdy \\ &+ \int_\Omega (f(x,y) - \mu_2)^2 (1 - H(\psi(x,y))) dxdy. \end{aligned} \tag{8.55}$$

At this point, we have an expression for the metric function, but it depends on a Heaviside function, which is not continuous, and its derivative that, while defined, has no numerical value. We must replace H with something that can be manipulated mathematically. Such an expression is called a *regularizer*. One such regularizer for H is the one used by

Chan and Vese [8.13]:

$$H_\epsilon(z) = \frac{1}{2}\left(1 + \frac{2}{\pi}\arctan\left(\frac{z}{\epsilon}\right)\right), \tag{8.56}$$

which in the limit as $\epsilon \to 0$ approaches H.

The parameters μ_1 and μ_2 can be written in terms of H as

$$\begin{aligned}\mu_1(\psi) &= \frac{\int_\Omega f(x,y)H(\psi(x,y)dxdy}{\int_\Omega H(\psi(x,y))dxdy}\\ \mu_2(\psi) &= \frac{\int_\Omega f(x,y)(1-H(\psi(x,y))dxdy}{\int_\Omega (1-H(\psi(x,y))dxdy}.\end{aligned} \tag{8.57}$$

Now that we have an expression for an objective function, we must derive a way to change ψ that will move the zero level set in the direction to minimize this function. This can be shown[6] to result in the time step equation, Eq. 8.58 below. To conserve space, the subscripts i and j respectively are used for the column and row numbers. The superscript n denotes the iteration number.

$$\begin{aligned}\psi_{i,j}^{n+1} \leftarrow \big[\psi_{i,j}^n + \Delta t\delta_\epsilon\big(\psi_{i,j}^n\big)\big(A_{i,j}\psi_{i+1,j}^n + A_{i-1,j}\psi_{i-1,j}^{n+1} + B_{i,j}\psi_{i,j+1}^n + B_{i,j-1}\psi_{i,j-1}^{n+1}\\ - (f_{i,j}-\mu_1)^2 + (f_{i,j}-\mu_2)^2\big)\big]/[1+\delta_\epsilon(\psi_{i,j})(A_{i,j}+A_{i-1,j}+B_{i,j}+B_{i,j-1})].\end{aligned} \tag{8.58}$$

Note that on the right-hand side, the term $\psi_{i-1,j}^{n+1}$ appears. This may be confusing since if this assignment statement is computing ψ^{n+1}, how can it use ψ^{n+1}? This equation describes an in-place modification of the ψ function, proceeding from left to right and top ($j = 0$) to bottom ($j = maxrow$). In this way, it is possible to use ψ^{n+1} on both the right and left of the assignment statement because on the right, pixel $\psi_{i,j}^{n+1}$ is being computed, and it depends on its already-computed neighbor $\psi_{i-1,j}^{n+1}$.

The per-iteration time difference Δt is usually chosen to be 1.

The A's and B's may be found by

$$\begin{aligned}A_{i,j} &= \frac{\alpha}{\eta + \big(\psi_{i+1,j}^n - \psi_{i,j}^n\big)^2 + \big(\big(\psi_{i,j+1}^n - \psi_{i,j-1}^{n+1}\big)/2\big)^2}\\ B_{i,j} &= \frac{\alpha}{\eta + \big(\big(\psi_{i+1,j}^n - \psi_{i-1,j}^{n+1}\big)/2\big)^2 + \big(\psi_{i,j}^n - \psi_{i+1,j}^n\big)^2}.\end{aligned} \tag{8.59}$$

The η is some small number used to avoid the possibility that the remainder of the denominator may be zero.

Figures 8.24 and 8.25 illustrate the use of the Chan-Vese region-based algorithm to segment the lion toe image and the sting ray image. In both cases, the algorithm is initialized by a contour far from the interest area.

The performance of Chan-Vese on the stingray image, is particularly interesting because it misses the edge on the lower right, which is really quite clear, *as an edge*. But Chan-Vese doesn't use edges, and in this case the color of the body of the ray is perfectly matched to the color of the sand, and the edge is missed, even though it would be clear to an edge

[6] `http://www.ipol.im/pub/art/2012/g-cv/article.pdf` presents a nice tutorial.

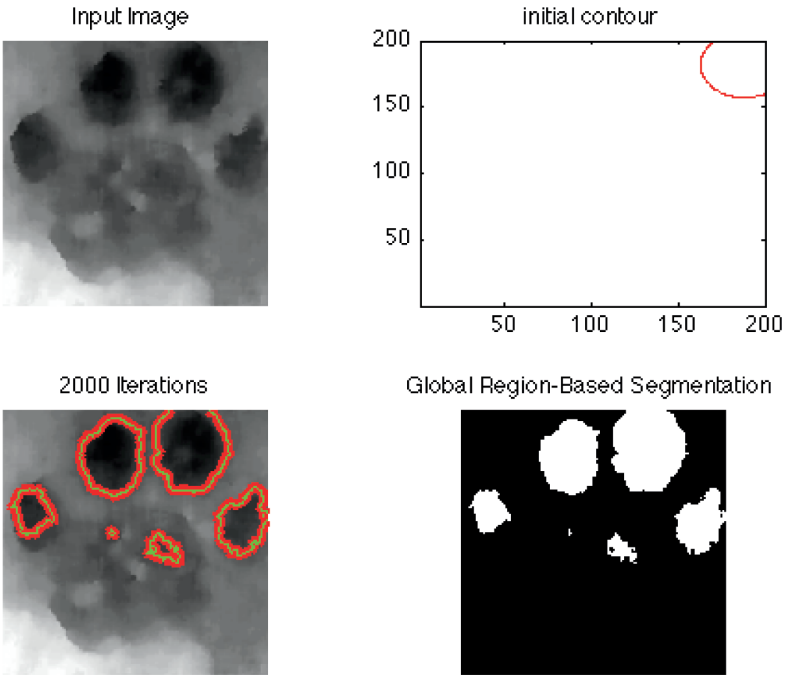


Figure 8.24 The original lion foootprint image, the initial contour, the edges found by Chan-Vese, and the resulting segmentation. Matlab code from `https://www.mathworks.com/matlabcentral/fileexchange/23445-chan-vese-active-contours-without-edges` was used.

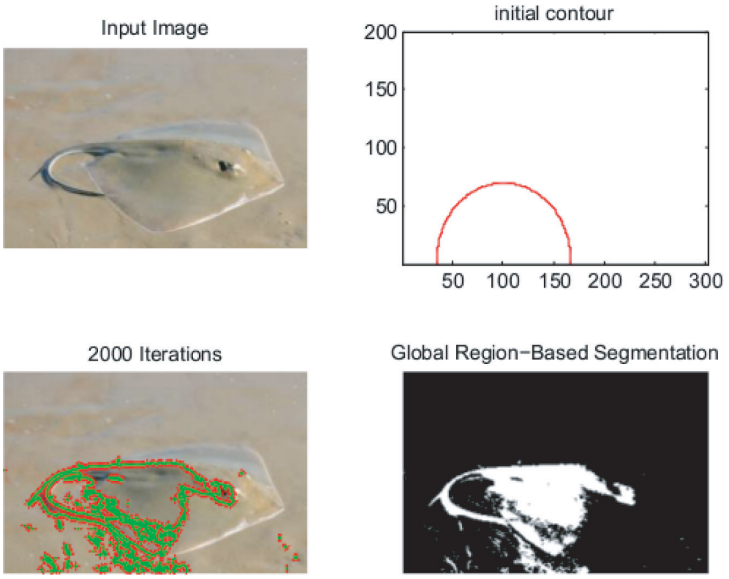


Figure 8.25 The original color image of a stingray in sand, the initial contour, the final contour found by Chan-Vese, and the resulting segmentation. Matlab code from `https://www.mathworks.com/matlabcentral/fileexchange/23445-chan-vese-active-contours-without-edges` was used.

53	52	51	53	52	51	53	50	51
49	50	**49**	**51**	40	41	39	41	40
48	**47**	12	12	**18**	19	16	15	20
46	**41**	12	12	**19**	20	17	15	16
45	**42**	12	**15**	18	17	19	17	18
46	44	**43**	44	41	16	18	20	19

Figure 8.26 An image with three drains marked with shading, including the 5-pixel drain with brightness of 12, the 2-pixel drain with brightness of 15, and the 1-pixel drain with brightness of 16.

detector. This emphasizes the need for something additional. Perhaps a hybrid approach that could combine information from edges with region differences?

The concept introduced by Chan and Vese of describing the foreground (or interior) and the background (or exterior) by statistics continues to be a powerful idea. For example, Ramudu et al. [8.40] follow the same philosophy, but introduce a force (which they call an *spf*) equal to

$$spf(f(x,y)) = \frac{f(x,y) - \frac{1}{2}(\mu_1 + \mu_2)}{\max(f(x,y) - \frac{1}{2}(\mu_1 + \mu_2))} \tag{8.60}$$

and show that using this force to move the contour often produces even better segmentations than Chan-Vese.

8.6 Watersheds: Segmentation Based on Brightness Surface

In this section, you will learn to think about moving up and down the brightness surface as water would move. We continue to consider the brightness function $f(x, y)$ as if f represented vertical distance, and the brightness image is thought of as a surface.

Suppose we pour water onto that surface. Water runs downhill, and (assuming it isn't absorbed or evaporated) will eventually come to rest at some place that is a minimum of local elevation. In a digital (sampled and quantized) image, such a minimum is not necessarily a single pixel. It could be a set of neighboring pixels, all of which have the same brightness. We call that final, lowest set of points a *drain*. In general, there are many drains in an image. Given a single point (x, y) on the surface, place a single drop of water on the surface at that point, and follow it until it comes to rest at some drain. The collection of all surface points that flow to a single drain, say drain i, is referred to as the *basin* or *catchment basin* associated with that drain.

Some surprising problems occur when attempting to use basins to segment images. First, from a single point, water will flow in the direction of steepest slope, but it is certainly possible that there might be equal, steepest slopes in more than one direction. So the problem of tracking where that one drop of water flows could be challenging and have high computational complexity. But, instead of following water as it flows down, let's instead start at the drain, and find all the neighboring points that definitely flow into that drain. That calculation can be done locally. Then, we find the points that drain into those points, etc.

Figure 8.26 illustrates pixel brightnesses in the vicinity of a 5-pixel drain. Each pixel "looks at" its neighbors and says, "If you are my neighbor and are brighter than me, you

53	52	51	53	52	51	53	50	51
49	50	49	51	40	41	39	41	40
48	47	12	12	18	19	16	15	20
46	41	12	12	19	20	17	15	16
45	42	12	15	18	17	19	17	18
46	44	43	44	41	16	18	20	19

Figure 8.27 Each pixel in the drain examines all of its neighbors (in this example, 4-connectedness is used). Any neighboring pixel that is brighter than the drain pixel is labeled as belonging to the same basin. The first drain to be processed is the darkest.

belong to my basin, unless you already belong to a different basin, but if you belong to a different basin already, you are a watershed pixel." This assigning of labels to pixels may remind one of connected components, but an important difference is that here, pixels are sorted by brightness before making a neighborhood decision.

The watershed-based segmentation problem may be cast as finding the basin to which each point in the image belongs.

As each basin fills with water, at some points the water level will be so high that water from two neighboring basins may mix. The locus of points along which this happens is called the *watershed lines* or just *watershed*.

This is illustrated in a simple example in Figures 8.26–8.29, showing steps in developing the watershed-based segmentation of a simple example image. This set of figures starts identifying three minima in the first image, which are the initial drains.

There are a variety of ways to compute watersheds on discrete, sampled data. Researchers have learned that watershed methods frequently must be adjusted slightly to comply with peculiarities of data. For example, a lot of noise will create thousands of tiny, uninteresting watersheds. Here, we present the approach provided in Algorithm 4.3 of [8.42]. We present the algorithm first, and then discuss a critical assumption. First, some definitions are presented:

A point p is *immediately downstream* of a point q if p and q are neighbors, and if the slope $\frac{f(q)-f(p)}{d(p,q)}$ is maximal over the set of neighbors. Here, $d(p, q)$ is the distance between p and q. If using 4-connected neighborhoods, d will always be 1.0. The set of points $p_q = \{p_1, p_2, \dots\}$, which are immediately downstream of q are denoted $\Gamma^{\downarrow}(q)$, and we use the notation $p \in \Gamma^{\downarrow}(q)$ to denote that p is immediately downstream of q. A point p_1 is *downstream* of p_n if there is a path of points $(p_1, p_2, \dots, p_n)$ where $p_i \in \Gamma^{\downarrow}(p_{i+1}) \forall i$.

53	52	51	53	52	51	53	50	51
49	50	49	51	40	41	39	41	40
48	47	12	12	18	19	16	15	20
46	41	12	12	19	20	17	15	16
45	42	12	15	18	17	19	17	18
46	44	43	44	41	16	18	20	19

Figure 8.28 Pixels are chosen in order of brightness, and the process of examining neighbors repeats. In this pass, several pixels are identified as belonging to two basins, and hence are marked (in orange) as watershed pixels.

53	52	51	53	52	51	53	50	51
49	50	49	51	40	41	39	41	40
48	47	12	12	18	19	16	15	20
46	41	12	12	19	20	17	15	16
45	42	12	15	18	17	19	17	18
46	44	43	44	41	16	18	20	19

Figure 8.29 One more pass identifies all the basins and watersheds.

We emphasize that $\Gamma^{\downarrow}(q)$ is a *set*. Water at q can (and often will) go in multiple directions.

A point q is *immediately upstream* of p if $p \in \Gamma^{\downarrow}(q)$, and the set of immediately upstream points of p will be

$$\Gamma^{\uparrow}(p) = \{q | p \in \Gamma^{\downarrow}(q)\}. \tag{8.61}$$

Examining Eq. 8.61, one observes that two searches are required to determine if $q \in \Gamma^{\uparrow}(p)$. First, find all the points q that are neighbors of p. Then, for each of those, determine if p is downstream of that q.

With that notation clarified, the segmentation algorithm is fairly straightforward:

1. Find all the drains in an image, and construct a label image (see section 8.1) in which all the labels of a particular drain are identical, and are distinct from all other labels.
2. Determine a structure containing every point in the image that is not the interior of a drain. Call that set V.
3. Search V to find a point with minimum brightness and call that point p; (initially, p will be an exterior point of a drain). Give p the label of that drain.
4. Remove p from V.
5. For all points $q \in \Gamma^{\uparrow}(p)$ that remain in V:
   (a) if q is not labeled, assign q the same label as p
   (b) if q is labeled and has a different label from p, give q a label indicating it is a pixel on the boundary of two basins (a watershed).
6. If V is not empty, go back to step 3.

Step 4 has the effect of making the search in step 3 increasingly faster as the process continues.

A problem arises in this algorithm if *plateaus* exist. Plateaus are local collections of adjacent pixels that are equal in brightness, but have boundary pixels both greater or lesser in brightness. Such regions might not be local minima, since they could have neighboring pixels lower in brightness. To allow the algorithm above to work properly, an image must be distorted in such a way as to make it *lower complete*. In a lower complete image, every non-minimum pixel has a lower neighbor. See [8.42] for an algorithm to make any image lower complete.

The observant student will notice that in Figures 8.26–8.29, there are no plateaus that are not minima. That is, this example was deliberately chosen to be a lower complete image.

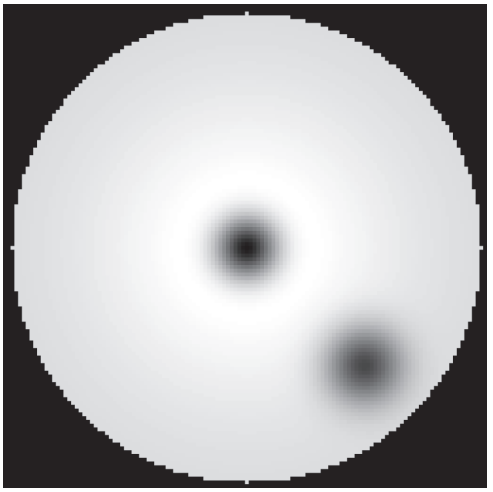


Figure 8.30 A "volcano-like" image that increases in brightness toward the middle, and then drops off suddenly. In addition, a second "volcanic vent" is on the lower right. Library name: ws1.png)

Figure 8.30 illustrates a synthetic image that grows brighter as one approaches the center. However, instead of coming to a peak, the brightness function rather suddenly drops significantly, forming a function rather like a volcano. On the side of the volcano, a second sudden minimum occurs (a "volcanic vent"). This brightness function was passed through a watershed program that identified which pixels belong to which basin, and identifying those basins with colors, forming Figure 8.31.

In order to use the philosophy of watersheds, or of many other Computer Vision techniques, one must be able to find minima in an image. We now consider this important component of watersheds.

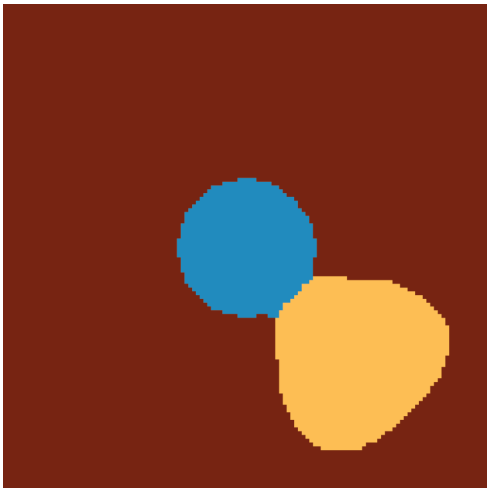


Figure 8.31 False coloring of the regions identified by applying a watershed algorithm to the adjacent figure.

Finding Minima

Here, we address a seemingly simple issue, finding local minima in an image. It would appear obvious: just look for pixels that have a lower value than all their neighbors. However, in a quantized image, it is possible for two adjacent pixels to have the same brightness. Thus, we must redefine a minimum as follows:

Definition: ***a minimum*** is a connected set of pixels, R, all with the same brightness, in which no pixel has any immediate neighbors of lesser brightness. That is,

$$R = \{p | f(p) \leq f(q), q \in \eta(p)\},$$

where $\eta(p)$ denotes the local neighborhood of p. $\eta(p)$ may be defined using 4- or 8-neighborhoods in 2D images and 6-, 8-, or 26-neighbors in 3D.

Given this definition, it would still seem simple to find minima. Just find all regions that are all the same brightness, but have outer boundary pixels that are minima of their immediate neighborhoods. Unfortunately, it isn't all that simple, because it requires a solution to the connected components problem.

Instead, let's think of it in sort of a brute-force way:

Start with two images, as in our discussion of connected components, a brightness image, f and a label image L. Initialize all elements of L to COULDBE, meaning that this pixel might be an element of a minimum. Then use the following pseudocode algorithm:

```
for (p = 0; p < NumberOfPixels)
       if (L(p) != NOTAMIN)
       {
              for q in the immediate neighborhood of p
              {
                     if (f(p) > f(q) ) L(p) <-- NOTAMIN
                     if (f(p) == f(q) && L(q) == NOTAMIN) L(p) <-- NOTAMIN
              }
       }
```

One pass of this algorithm produces a label image that has pixels marked as "Cannot be a Minimum" (NOTAMIN) or as "unknown" (COULDBE).

A few iterations of the same algorithm, applied to successive label images, will converge to real minima.

The surprising part of this extremely simple algorithm is how quickly it converges. The iterations stop as soon as no pixels are relabeled from COULDBE to NOTAMIN. The graph in Figure 8.32 shows the number of pixels marked as COULDBE during multiple iterations of the algorithm running on (blue) a simple 128×128 image, and (red) a 141×141 image with a great deal more complexity (a lion footprint in sand). From the figure, we observe that finding all minima in a simple 128×128 image requires only four passes through the data, and a very complex, noisy 141×141 image with a lot of clutter requires only ten passes.

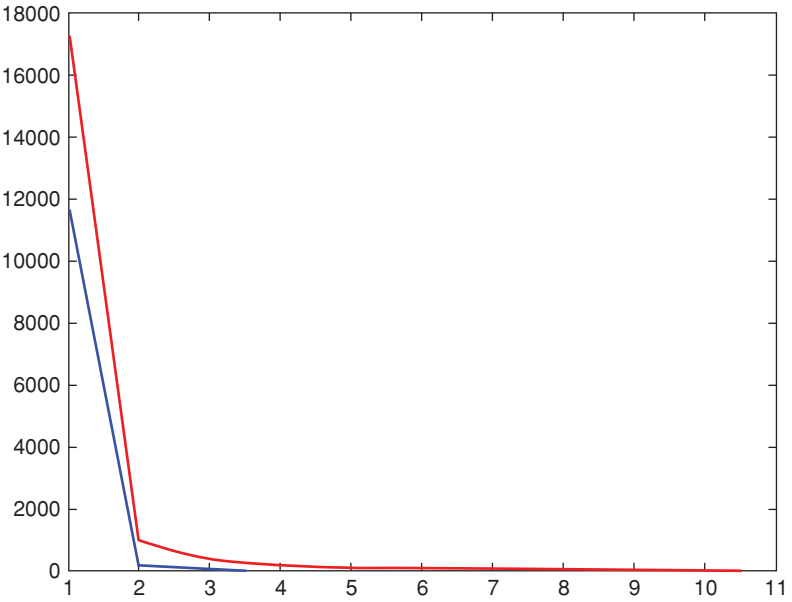


Figure 8.32 The convergence of finding minimum regions in two images.

8.6.1 Watershed Example

In this subsection, we illustrate an example image for which the watershed philosophy is not well suited, and explain why.

Figure 8.33 illustrates a range image of a lion footprint on sand. Darker pixels are further from the camera. As the animal raises her foot after stepping, the motion of the foot kicks sand into the footprint, creating an image that has noise as well as a substantial amount of random clutter. Such an image has a great many minima, as illustrated in Figure 8.33b. (The student is reminded of the definition presented above: A minimum is a

(a) Original image. Library name: ft512B.png

(b) This image has 206 minima. The minima are randomly colored in this illustration, with green denoting background.

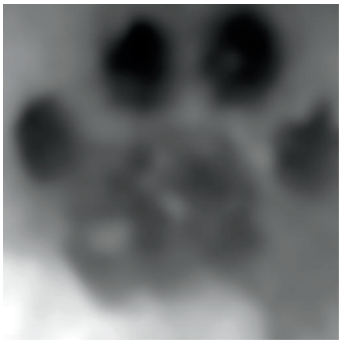
(c) Watershed segmentation resulting from using these 206 minima.

Figure 8.33 A very noisy image with 206 minima, produces a meaningless watershed segmentation.

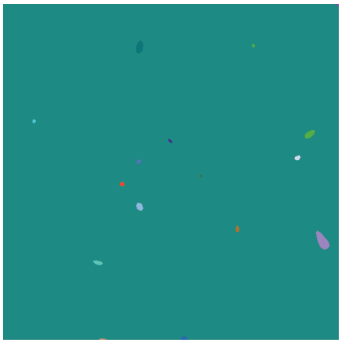
(a) Image blurred with a $\sigma = 3$ Gaussian.

(b) This image has 73 minima.

(c) Watershed segmentation resulting from using these 73 minima



(d) Image blurred with a $\sigma = 9$ Gaussian.



(e) This image has 17 minima.

(f) Watershed segmentation resulting from using these 17 minima.

Figure 8.34 After more blurring, the number of minima is reduced to 17, and a meaningful watershed segmentation can be extracted. The four regions corresponding to the four toes are marked with numbers. The boundary of the upper sides of the toes is not distinguished because these edges do not comply with the assumptions of the watershed philosophy: a ridge between two basins.

connected set of pixels, all of which are the same brightness, and none of which have a darker
neighbor.)

There are a variety of ways to approach random clutter and noise like this, but the easiest is blurring. Figure 8.34a illustrates a blurred version of the original. The blurring reduces the number of minima to 73. Not enough, however, to produce a meaningful watershed segmentation.

Finally, as illustrated in Figure 8.34, after much more blurring, an image is produced that has only 17 minima. Regions corresponding to the four toes are easily identified, and the lower boundaries of those toes align with the toes as a human would segment them. The upper boundary of the toes, however, is still not identified, because the background slopes upward, away from the toes all the way to the image boundary, and there is no watershed line.

There are ways to make watersheds work better. For example, erroneous segments may be simply eliminated by removing the initializing minimum corresponding to that region. The student is referred to [8.6, 8.5, 8.42] to list just a few relevant references.

The point of this example is to enforce the concept that one must choose a segmentation algorithm appropriate to the problem at hand. For example, in sections 8.7–8.8 different tools will be shown to be well suited to this application.

8.7 Graph Cuts: Segmentation Based on Graph Theory

In this section, you will discover another reason for knowing the literature in graph theory. If one can determine an objective function suitably defined on an image graph, then the graph theory literature provides algorithms for optimizing this objective function.

Represent the image by a graph (the *image graph* of Section 4.2.5), in which the pixels are the nodes and edges exist between nodes if the pixels are neighbors. A *weight* w_{ij} is assigned to the edge between nodes i and j. The weights could be scalars that measure how similar the two nodes are. Let S_1 and S_2 be two sets of nodes. A *cut* between S_1 and S_2 is a collection of edges that, if removed, would separate the two sets into separate components of the graph, that is $S_1 \cap S_2 = \emptyset$.

The weight of the cut, $cut(S_1, S_2)$ is the sum of the weights of the edges that are cut.

To find an optimal cut, we might seek to find the cut with minimal weight. However, this simple concept won't work in general without modification because it could produce cuts surrounding a single pixel. Instead, some sort of normalization is needed:

Let the entire graph be denoted by $V = S_1 \cup S_2$, with the understanding that sets S_1 and S_2 are disjoint ($S_1 \cap S_2 = \emptyset$). Then a *normalized cut*, can be defined by an objective function

$$N(S_1, S_2) = \frac{cut(S_1, S_2)}{W_1 + cut(S_1, S_2)} + \frac{cut(S_1, S_2)}{W_2 + cut(S_1, S_2)} \tag{8.62}$$

where W_1 is the sum of all the weights within region S_1, $W_1 = \sum w_{ij}, i \in S_1, j \in S_1$. W_2 is similarly defined.

Now, the problem: finding the optimal normalized cut is in general an NP-complete problem. That is, there is no algorithm currently known that can find the optimal cut in less than exponential time. There are many approaches to this problem [8.54, 8.46, 8.36, 8.52].

To illustrate the idea, we will summarize the approach of Boykov and Funka-Lea [8.8]. This method uses graph cuts to find the optimal foreground-background segmentation of an image. First, the image graph described in section 4.2.5 is augmented by adding two "terminal" nodes, s and t. Any pixel with a connection to s will be considered foreground and any pixel with connection to t will be considered background. Initially, both connections will be there. The graph is augmented to change the set of nodes V to $v = V \cup \{s, t\}$ and the set of edges $E = \{(p_i, p_j)\}$ is augmented to become $\xi = E \cup \{(p_i, s), (p_i, t)\}\forall_i$. A pictorial representation of this augmented graph after a cut is given in Figure 8.35. It will be convenient to also define S as the set of pixels with an edge to s and T as the set of pixels with an edge to t. Initially, $S = T$, and the algorithm will cut edges until $S \cap T = \emptyset$.

8.7.1 An Objective Function

Use of this graph for segmentation requires a complete cut. Such a cut C is the set of all edges that are cut, including edges between pixels and edges between pixels and terminals. A complete cut must have each pixel be connected to either s or t, but not to both. The

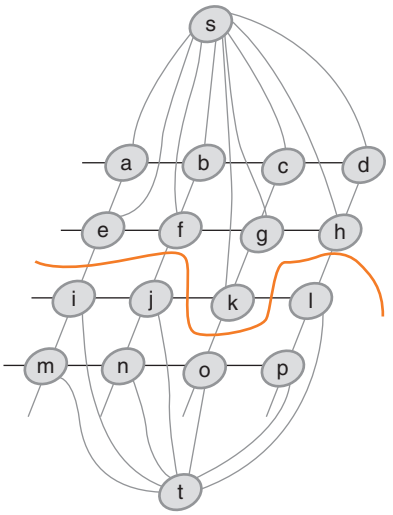


Figure 8.35 An image graph augmented with two labeling nodes, s and t. Some pixels are connected to the terminal node s, and some to t. The cut $\{(e, i), (f, j), (j, k), (k, o), (k, l)\}$ completely divides the pixels. For simplicity, cuts of the edges to the terminals, such as (c, t) are not shown, but such edges are actually part of the cut.

trick, of course, is how to find the optimal such cut. Two issues must be resolved: first, "what does *optimal* mean here?" and second, "how do we find the best such cut?"

An objective function will be proposed below, following [8.8]. For this, we define a set of *costs*, penalties for bad decisions. We then seek to make good decisions by minimizing the total cost.

The objective function first codifies what we know about the two regions, s and t, (object and background). Presuming we have some information about object pixels and background pixels, we could describe what we know about a pixel assuming one of those two choices by using conditional probabilities. That is, we could use the probability that pixel p has brightness f_p given that it belongs to foreground, $Pr(f_p \mid foreground)$ and similarly $Pr(f_p \mid background)$. We must modify this objective slightly, since we are seeking a minimum at the best choice, and the probability has a maximum at the best choice, so we might try a negative log-likelihood, such as

$$\begin{aligned} R_p(foreground) &= -\ln Pr(f_p \mid foreground) \\ R_p(background) &= -\ln Pr(f_p \mid background). \end{aligned} \tag{8.63}$$

In addition to how well each pixel matches its assigned region, the objective function must also codify the differences between pixels. Define $B(p, q)$ to be the cost of making an error in deciding to cut (or not cut) an edge. If there is a strong brightness difference between those two pixels, and we choose to not cut, the penalty should be high.

Similarly, a high penalty should result if the two have similar brightnesses but we choose to cut. One representation of similarity would be our familiar function:

$$B(p, q) = \exp\left(-\frac{(f_p - f_q)^2}{2\sigma^2}\right), \tag{8.64}$$

where σ is some measure of image noise. This is large (approaching 1.0) when the brightnesses are similar, and we should NOT cut this edge.

In addition to edges between pixels, we must consider edges between a pixel and a terminal node, the cost of (p, s) and (p, t). Remembering that we are considering the cost of a bad decision, simply use $R_p(background)$ for the cost of (p, s) and $R_p(foreground)$ for the cost of (p, t).

To arrive at a single scalar to use as the objective function, define a vector A (for "assignment") with individual elements A_i, one such element for each pixel. Each element of A, A_p, determines a binary assignment for pixel p:

$$A_p = \begin{cases} 0 & background \\ 1 & foreground \end{cases}$$

and from Eq. 8.64, we can interpret B as meaning:

$$B = \begin{cases} 0 & \text{if the two pixels have very different brightnesses.} \\ 1 & \text{if the two pixels have very similar brightnesses.} \end{cases}$$

Thus, we seek the value of A that minimizes

$$E(A) = \sum_p R_p(A_p) + \sum_{p,q} \delta(p, q) B(p, q) \tag{8.65}$$

where $\delta(p, q) = \begin{cases} 1 & \text{if } A_p \neq A_q \\ 0 & \text{if } A_p = A_q \end{cases}$. If the two pixels are labeled differently ($\delta = 1$), then they contribute to the error if their brightnesses are similar. The function $\delta : Z \times Z \rightarrow \mathbb{R}$ incorporates our intuition that edges that join pixels with large differences in brightness should be edge pixels.

8.7.2 Solving the Objective Function

Minimization of this objective function is a combinatorial optimization problem, in which we must find the value of A that makes E minimal.

The expression *feasible cut* is defined as follows: A cut is feasible if (1) for every pixel, the cut severs either the link to s or t, but not both; and (2) for each edge (p, q) that is cut, p and q have links to different terminals.

It is possible to relate the process of finding the optimal $\hat{A} = argmin_A E(A)$ to the problem of finding the optimal feasible cut $\hat{C}$. Specifically, it is possible to show that the labeling of pixels (the value of A_p) resulting from the optimal cut will be the optimal labeling.

Thus instead of finding the optimal labeling by combinatorial search, we can seek the optimal cut of the graph and get the same result. There are a number of algorithms in the graph theory literature for finding optimal cuts. These are discussed and evaluated in [8.9]. We will not pursue this further here because it will lead us away from Computer Vision and into Graph Theory.[7]

[7] It's important though; consider taking a course in graph theory.

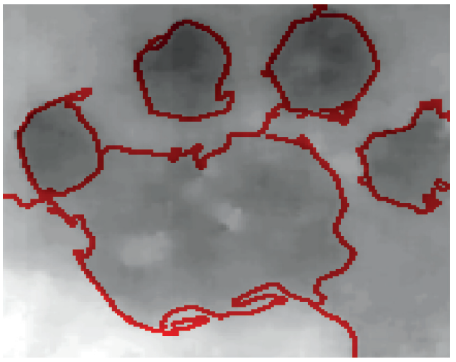


Figure 8.36 Graph cuts can also be used to find a segmentation of the very noisy lion toe image. The resulting image has identified the toes quite well, but has introduced some extraneous edges. These edges require post-processing.

Figure 8.36 illustrates the result of using this particular objective function and graph cut optimization to segment the lion toe image.[8]

8.8 Segmentation Using MFA

In section 6.5.2 you learned how to apply the mean field annealing (MFA) algorithm to recover an image that

1. resembles the original image and
2. has regions that are uniform in brightness but have sharp edges. We called that "piecewise-constant."

Suppose we use the piecewise-constant prior of that section, and run the MFA algorithm until the temperature is *very* low. Recall that as temperature gets low, the solution approaches more and more closely the best solution that satisfies the prior. Thus, by the simple strategy of running the MFA algorithm long enough, we get a binary solution – a segmentation.

Figure 8.37 illustrates the application of this method to segmentation of a very noisy, blurred image with few sharp, discernible edges. The four toes of the lion are segmented into areas of uniform brightness that are then labeled using connected components to identify them using different colors.

8.9 Evaluating the Quality of a Segmentation

In this section you will learn why no single segmentation works well for all applications, and you will learn a few ideas about how to compare competing algorithms with the same application context.

[8] The authors are grateful to Zhifei Zhang for his assistance in producing this result.

(a) Original image, a range image of a lion footprint where brighter means deeper. Library name:fourtoesoriginal.png

(b) Image segmented using a piecewise-constant MFA prior.

Figure 8.37 Using MFA to segment a range image of a lion footprint.

As you have concluded by now, there are many algorithms and variations on algorithms for segmentation. But which is the best? We need an algorithm to evaluate the quality of segmentations. But which is the best such evaluation algorithm? We need an algorithm that evaluates the quality of evaluation algorithms that . . . (help! We are getting into the halting problem!).

Nevertheless, there are several approaches to evaluating segmentation quality. Since one result of a segmentation is edges, you could indirectly infer segmentation quality by measuring edge positions. Pratt [8.39] provides one such algorithm.

Hoover et al. [8.25] proposed the following formalism for comparing the quality of a machine-segmented (MS) image using a human-segmented ground-truth (GT) image as the "gold standard." Let M and G denote the MS and GT images respectively; let M_i ($i = 1, \ldots, m$) denote a patch in M; and let G_j ($j = 1, \ldots, m$) denote a region in G. $|R|$ will denote the number of pixels in region R. Let O_{ij} be the number of pixels that belong to both patch i in the MS image and region j in the GT image. Finally, let T be a threshold, $0.5 < T \leq 1.0$.

There are five different segmentation results that may be obtained, only one of which is correct. They are defined as follows:

1. A *correct* classification occurs when $O_{ij} \geq T\,|M|$ and $O_{ij} \geq T\,|G|$.
2. *Oversegmentation* occurs when a region in the GT image is broken into several patches in the MS image.
3. *Undersegmentation* occurs when pixels in distinct regions in the GT image are identified as belonging to the same patch in the MS image. The definition is isomorphic to the definition of oversegmentation with the two images reversed.

4. A *missed classification* occurs when a region in the GT image is neither correctly segmented nor is part of an oversegmentation or an undersegmentation.
5. A *noise classification* is identical to a missed classification except that the region is really a patch in the MS image.

In the case of range imagery, two segmentations can be further compared by computing the normal vectors to the GT region and the corresponding MS patch and then finding the absolute value of the angle between these two vectors. With these definitions, we can evaluate the quality of a segmentation of a range image by counting correct or erroneous segmentations, and measuring total angle error. By plotting these measures vs. T and comparing these plots, a measure of segmenter performance may be determined.

Hoover et al. [8.25] used this approach to do a thorough evaluation of the quality of four different range image segmentation algorithms.

A simpler algorithm is the Jaccard similarity index [8.27]: Let S_1 be the area of the region returned by the segmentation process, and let S_2 be the same measure returned by a human expert. Then

$$J(S_1, S_2) = \frac{S_1 \cap S_2}{S_1 \cup S_2}. \tag{8.66}$$

At the International Conference on Computer Vision and Pattern Recognition (CVPR), 2004, David Martin [8.35] presented an excellent tutorial entitled "Evaluating Segmentation," which was primarily focussed on graph-based algorithms, but is also a good tutorial on evaluating segmentation quality.

8.10 Conclusion

In this chapter, we used the concepts of consistency to identify the components of a region. In section 8.2, if all pixels are the same brightness, they are defined to be in the same region. Some additional processing is generally needed when analyzing histograms (Chow and Kaneko [8.17]; Rosenfeld and Kak [8.43]. Trier and Jain [8.53] explain and experimentally compare several such methods).

The use of various optimization methods is a dominant theme in this chapter. For example, in section 8.2.2 we used an optimization method, minimum squared error, to find the best threshold. In section 8.5, we obtained a closed boundary using the philosophy of active contours, by specifying a problem-specific objective function and finding the boundary that minimizes that function. Any appropriate minimization technique could be used.

So what does one use? Of course, the answer is always "it depends." It depends on your problem.

If you have a problem in which the data is already binarized (partitioned into background and foreground), and you need to know which pixels belong to which object, then connected components is the tool of choice. In that case, if you have one (x, y) pair that you know to be in the region of interest, you may use region growing starting at that point; or if you happen to have a version of the iterative connected components algorithm that is easy to use (as the authors do), just run it on the entire image (it won't take long), and then see which region contains the pixel of interest.

If you don't already have the foreground and background defined, you can use connected components to do segmentation by setting a local difference (the R of Eq. 8.19) and not allowing two pixels with greater difference than this to be in the same region. However, since this is a local operation, it is subject to "leaking." That is, if a region changes brightness sufficiently slowly, background and foreground may be inadvertently merged. If, however, $R(p, q)$ is defined to require equal brightness, then connected components will produce a segmentation. Leaking can be prevented by simply defining

$$R(p,q) = \begin{cases} TRUE & if \quad |f(p) - f(q)| < \text{a sufficiently small threshold} \\ FALSE & \text{otherwise.} \end{cases} \tag{8.67}$$

The snake problem fits perfectly into the MAP philosophy, and simulated annealing (SA) can be used [8.50]. However, the search neighborhood is problematic. That is, as we have discussed, SA is guaranteed to find the state that globally minimizes a set of states. However, the set of states must be sampled in order for SA to work. In [8.50], an existing contour is used as a starting point, and at each iteration, the only states sampled are contours within one pixel of the current contour, and from that set a minimum is chosen. The resultant contour is the best contour of the set sampled, but not necessarily the best contour from the entire region of interest.

The minimization problem for snakes in two dimensions is solvable using dynamic programming [8.24].

Baswarah et al. [8.3] provides a reasonable literature survey of the literature in Active Contours.

The idea of using level sets for adaptive contours was first proposed by Sethian [8.44, 8.45]. Malladi [8.34] extended this by observing that there are advantages to considering only a set of points near the current contour. Taubin [8.51] uses the concept of a level set implicitly in fitting piecewise-linear curves. Kimmel, Amir, and Bruckstein [8.28] demonstrate that level sets may be used for other things, such as finding the shortest path on a surface.

Not all algorithms that use a deformable contour philosophy follow the strategies described in section 8.5. For example, Lai and Chin [8.30] describe a variation that treats the contour points as a sequence of random variables, which may therefore be described by a Markov process, and optimized using MAP strategies.

Watersheds are interesting, and can be used to produce complete segmentations with different basins identified.

The interested reader may refer to an excellent survey/tutorial on watersheds by Roerdink and Meijster [8.42] for additional details. Another excellent tutorial is available on the Web at `http://cmm.ensmp.fr/~beucher/wtshed.html`.

Graph cuts find optimal segmentations. In principle, these will be superior to the cuts produced by connected components, since the user has control of the objective function and the optimization is global. However, some versions of graph cuts require prior information defining foreground and background. Therefore, they cannot segment images with homogeneous regions of different brightness, which using connected components and a very small threshold in Eq. 8.67 can accomplish.

8.11 Assignments

Assignment 8.1: The histogram of an image is defined and discussed in section 8.2.2. In that section, the histogram plays a significant part in identifying regions to be segmented. But what if the histogram were flat? Take the image assigned by your instructor and compute its histogram. Is it easy from this histogram to find the best dark/bright threshold? Discuss that. Find some way to run histogram equalization. That program will produce a histogram very close to flat. Consider the histogram of that image. Discuss.

Assignment 8.2: In Otsu's method, prove the statement that minimizing the intra-class variance is equivalent to maximizing the inter-class variance.

Assignment 8.3: In Otsu's method, derive the between-class variance expression in Eq. 8.3 and explain how it can be recursively calculated to speed up the procedure.

Assignment 8.4: Derive Eq. 8.41.

Assignment 8.5: You have a set of pixels that are drawn from an image that you know is black and white, but the paper is old and dirty, and the histogram clearly has two peaks, with lots of noise. You have an idea: consider only points near the two peaks. That would give you two independent histograms, and you could fit each of those with just 3 parameters (or . . . . you decide). Try this. You do not need to write software, just think about the question. Find the two peaks independently. Do you encounter any problems?

Assignment 8.6: Here are 6 points in a 3-space: [1, 2, 0], [2, 7, 2], [3, 4, 1], [7, 7, 6], [5, 6, 4], [0, 0, 1]. Start the k-means algorithm with 2 clusters located at [5, 6, 0] and [7, 1, 6]. Run two iterations of the algorithm using Euclidean distance. Show the output at each iteration. Repeat with Manhattan distance. What do you conclude? You may do it by hand, write a simple program, or just run some program that you find.

Assignment 8.7: Equation 8.4 presents the equation for the sum of two Gaussians. Consider the problem of estimating the six parameters:

1. Write the equation for the sum-squared error between the model of Eq. 8.4 and an actual histogram of n points.
2. Differentiate that equation with respect to A_1 and write the equation that needs to be solved to use the strategy of "take the derivative and set it equal to zero."
3. Differentiate that equation with respect to A_2 and write the equation that needs to be solved to use the strategy of "take the derivative and set it equal to zero."
4. Discuss the feasibility of solving all six equations simultaneously and suggest an appropriate approach to this problem.

Assignment 8.8: A very small image is illustrated in Figure 8.26. Suppose the pixels marked in **boldface font** are edge pixels, and the values given for all the pixels may be ignored. Find the distance transform value for all remaining pixels. Use both Euclidean and 4-connected distances.

Assignment 8.9: On color clustering. In your images directory are three images, called `facered.ifs`, `faceblue.ifs`, and (can you guess?) `facegreen.ifs`. These are the red,

blue, and green components of a full color image. Each pixel can be represented by 8 bits of red, 8 of green and 8 of blue. Therefore, there are potentially, 2^{24} colors in this image. Unfortunately, your computer only has 8 bits of color, for a total of 256 possible colors. Your mission (should you choose to accept it) is to figure out a way to display this picture in full color on your workstation.

Approach: use some kind of clustering algorithm. Find 128 clusters that best represent the color space, and assign all points to one of those colors. Then, make a file with the following data in it. For example:

brightness value	red	green	blue
1	214	9	3

This means that if a pixel has brightness 1, it should be displayed on the screen as 214 of red, 9 of green and 3 of blue. Such a pixel will show up as almost pure red. Thus, each cluster center is represented by a color. In the example above, cluster center 1 is the almost pure red point. Now, make an image in which every pixel has the brightness equal to the cluster number of the nearest cluster.

Repeat the same experiment with 64, 32, 16, 8, and 4 clusters. Compare the difference.

Assignment 8.10: Use finite differences discussed in section 8.5.1 to estimate the fourth derivative based on both the symmetric (Eq. 8.68) and asymmetric (Eq. 8.69) definitions of derivatives. Compare the differences.

$$\frac{\partial u}{\partial s} = \lim_{\Delta s \to 0} \frac{u(s + \Delta s) - u(s - \Delta s)}{2\Delta s} \tag{8.68}$$

$$\frac{\partial u}{\partial s} = \lim_{\Delta s \to 0} \frac{u(s + \Delta s) - u(s)}{\Delta s} \tag{8.69}$$

Bibliography

[8.1] N. Badshah and K. Chen. Multigrid methods for the Chan-Vese model in variational segmentation. *Communications in Computational Physics*, 4(2), 2008.

[8.2] E. Bae and X. Tai. Efficient global minimization for the multiphase Chan-Vese model of image segmentation. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*. Springer, 2009.

[8.3] D. Baswaraj, A. Govardhan, and P. Premchand. Active contours and image segmentation: The current state of the art. *Global Journal of Computer Sciencce and Technology: Graphics and Vision*, 12(11), 2012.

[8.4] S. Beucher. Watersheds of functions and picture segmentation. In *IEEE ICASSP Conf.*, May 1982.

[8.5] S. Beucher and F. Meyer. The morphological approach to segmentation: The watershed transformation. In E. R. Dougherty, editor, *Mathematical Morphology in Image Processing*, pages 433–481. 1993.

[8.6] W. Bieniecki. Oversegmentation avoidance in watershed-based algorithms for color images. In *Proceedings of the International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science*, 2004.

[8.7] G. Bilbro and W. Snyder. Optimization of functions with many minima. *IEEE Transactions on SMC*, 21(4), July/August 1991.

[8.8] Y. Boykov and G. Funka-Lea. Graph cuts and efficient n-d image segmentation. *International Journal of Computer Vision*, 70(2), 2006.

[8.9] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. and Machine Intel.*, 26(9), Sept 2004.

[8.10] R. Boyles. On the convergence of the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 45(1), 1983.

[8.11] C. R. Brice and C. L. Fennema. Scene analysis using regions. *Artificial Intelligence*, 1, 1970.

[8.12] T. Chan, B. Sandberg, and L.Vese. Active contours without edges for vector-values images. *Journal of Visual Communication and Image Representation*, 11, 2000.

[8.13] T. Chan and L. Vese. Active contours without edges. *IEEE Transactions on Image Processing*, 10(1), 2001.

[8.14] T. F. Chen. Medical image segmentation using level sets. Technical Report TR CS-2008-12, University of Waterloo, 2008.

[8.15] Y. Chen and M. Gupta. Theory and use of the EM algorithm. *Foundations and Trends in Signal Processing*, 4(3), 2010.

[8.16] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(8), 1995.

[8.17] C. Chow and T. Keneko. Automatic detection of the left ventricle from cineangiograms. *Computers and Biomedical Research*, 5, 1972.

[8.18] G. Chung and L. Vese. Energy minimization based segmentation and denoising using a multilayer level set approach. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, volume 3757/2005, 2005.

[8.19] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. and Machine Intel.*, 24(5), May 2002.

[8.20] X. Dai, W. Snyder, G. Bilbro, R. Williams, and R. Cowan. Left-ventricle boundary detection from nuclear medicine images. *Journal of Digital Imaging*, February 1998.

[8.21] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1), 1977.

[8.22] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[8.23] K. Fukunaga and L. D. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Trans. on Information Theory*, IT-21(1), January 1975.

[8.24] D. Geiger, A. Gupta, L. Costa, and J. Vlontzos. Dynamic programming for Detecting, tracking, and matching deformable contours. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(3), March 1995.

[8.25] A. Hoover, G. Jean-Baptiste, X. Jiang, P. Flynn, H. Bunke, D. Goldgof, K. Bowyer, D. Eggbert, A. Fitzgibbon, and R. Fisher. An experimental comparison of range image segmentation algorithms. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(7), 1996.

[8.26] J. Ivins and J. Porrill. Everything you always wanted to know about snakes (but were afraid to ask). *AIVRU Technical memo 86*, july 2000.

[8.27] P. Jaccard. Distribution dela flore alpine dans le bassin des drouces et dans quelques regions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37(140), 1901.

[8.28] R. Kimmel, A. Amir, and A. Bruckstein. Finding shortest paths on surfaces using level sets propagation. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(6), June 1995.

[8.29] R. Kashyap and R. Chellappa. Estimation and Choice of Neighbors in Spatial Interaction Model of Images. IEEE Trans. Information Theory, IT-29, January 1983.

[8.30] K. Lai and R. Chin. Deformable contours: Modeling and extraction. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(11), November 1995.

[8.31] K. Lee, P. Meer, and R. Park. Robust adaptive segmentation of range images. *IEEE Trans. Pattern Anal. and Machine Intel.*, 20(2), 1998.

[8.32] S. Lloyd. Least square quantization in PCM. *Bell Systems Laboratories Paper*, 1957.

[8.33] S. Lloyd. Least square quantization in PCM. *IEEE Transactions on Information Theory*, 28(2), 1982.

[8.34] R. Malladi, J. Sethian, and B. Vemuri. Shape modeling with front propagation: A level set approach. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(2):158–175, February 1995.

[8.35] D. Martin. Evaluating segmentation. In *CVPR tutorial*, 2004.

[8.36] A. Ng, M. Jordan, and Y. Weiss. *On Spectral Clustering, Analysis and an Algorithm*. MIT Press, 2001.

[8.37] S. Osher and J. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79, 1988.

[8.38] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Trans. Sys., Man., Cyber.*, 9(1), September 1979.

[8.39] W. K. Pratt. *Digital Image Processing*. Wiley, 1978.

[8.40] K. Ramudu, G. Reddy, A. Srinivas, and T. Krishna. Global region based segmentation of satellite and medical imagery with active contours and level set evolution on noisy images. *International Journal of Applied Physics and Mathematics*, 2(6), 2012.

[8.41] R. Redner and H. Walker. Mixture densities, maximum likelihood, and the EM algorithm. *SIAM Review*, 26, April 1984.

[8.42] J. Roerdink and A. Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *Fundamenta Informaticae*, 41, 2001.

[8.43] A. Rosenfeld and A. Kak. *Digital Picture Processing*. Academic Press, 2 edition, 1997.

[8.44] J. Sethian. Curvature and evolution of fronts. *Comm. in Math. Physics*, 101, 1985.

[8.45] J. Sethian. Numerical algorithms for propagating interfaces: Hamilton-Jacobi equations and conservation laws. *Journal of Differential Geometry*, 31, 1990.

[8.46] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. and Machine Intel.*, 22(9), August 2000.

[8.47] W. Snyder and G. Bilbro. Segmentation of range images. In *Int. Conference on Robotics and Automation*, March 1985.

[8.48] W. Snyder and A. Cowart. An iterative approach to region growing using associative memories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, May 1983.

[8.49] W. Snyder and C. Savage. Content-addressable read-write memories for image analysis. *IEEE Transactions on Computers*, October 1982.

[8.50] G. Storvik. A Bayesian approach to dynamic contours through stochastic sampling and simulated annealing. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(10), October 1994.

[8.51] G. Taubin and R. Ronfard. Implicit simplicial models for adaptive curve reconstruction. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(3), March 1996.

[8.52] D. Tolliver and G. L. Miller. Spectral rounding with applications in image segmentation and clustering. In *International Conference on Computer Vision and Pattern Recognition*, 2006.

[8.53] O. Trier and A. Jain. Goal-directed evaluation of binarization methods. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(12), 1995.

[8.54] Y. Weiss. Segmentation using eigenvectors: a unifying view. In *Proceedings of International Conference on Computer Vision*, 1999.

[8.55] C. Wu. On the convergence properties of the EM algorithm. *Annals of Statistics*, 11, March 1983.

[8.56] S. Zhu and A. Yuille. Region competition: Unifying snakes, region growing, and Bayes/MDL for multiband image segmentation. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(9), 1996.

9 Parametric Transforms

Supposing I was on the other side of the glass, wouldn't the orange still be in my right hand?
– Lewis Carroll

9.1 Introduction

Suppose you are tasked with the problem of finding the straight lines in the image shown in Figure 9.1. If only one straight line were present in the image, straight line fitting could be used to determine the parameters of the curve. But there are two line segments here. If we could segment this image first, then we could fit each segment separately – yes, this is a segmentation problem, of a sort.

Another approach to the solution of the problem of inferring large-scale properties from local measurements is the use of parametric transformations. In this approach, we assume that the object for which we are searching in the image may be described by a mathematical expression, which in turn is represented by a set of parameters. For example, a straight line may be written in slope-intercept form:

$$y = ax + b, \tag{9.1}$$

where a and b are the parameters describing the line and relating the variables x and y. Our approach is as follows: given a set of points (or other features), all of which satisfy the same equation, we will find the parameters of that equation. In a sense, this is the same as fitting a curve to a set of points, but as we will discover, the parametric transform approach allows us to find multiple curves, without knowing *a priori* which point belongs to which curve. That is, *parametric transforms reverse the roles of variables and parameters*.

In this chapter, we begin this study by considering the special cases of finding straight lines, circles, and ellipses. We then extend the discussion to finding arbitrary shapes:

- (Section 9.2) The first such technique, the Hough transform, identifies separate straight lines in images. This section also discusses the computational complexity issue of this approach. Parametric transforms can take considerable time because, for every point of interest, a function must be plotted. However, it is possible to speed up this process by making use of image information such as image gradients. That is, we can trade off computation in image space for computation in parameter space.
- (Section 9.3) The search for circles in noisy images allows us to discover that functions with more than two parameters can be handled in a variety of ways, including simply increasing the dimensionality of the problem.

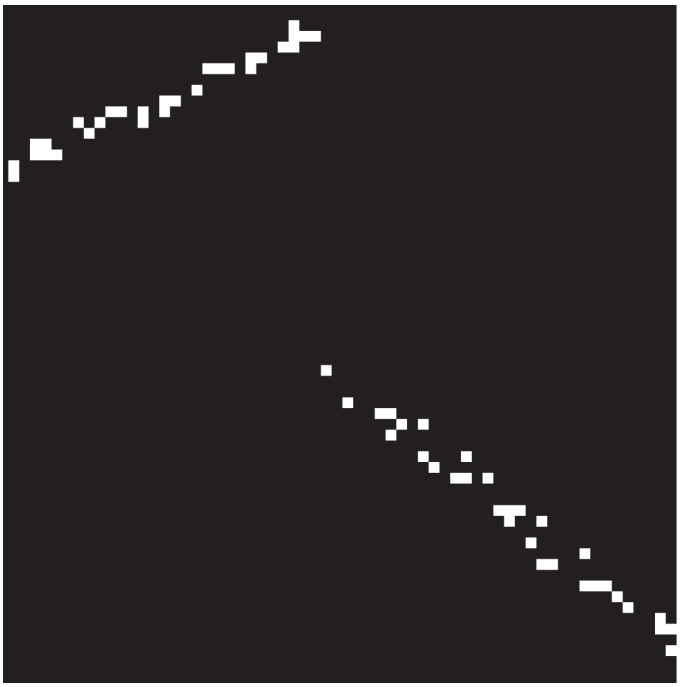


Figure 9.1 An image that is the output from an edge detector. The human can immediately discern that this boundary consists of two straight segments. Library name: work.png

- (Section 9.4) An ellipse in arbitrary orientation has six parameters, and following the simple parametric transform philosophy would require a six-dimensional search space. However, this section illustrates how to use geometric relationships to find ellipses using a much smaller search space.
- (Section 9.5) The Generalized Hough Transform illustrates methods for using the philosophy of this section to recognize shapes that are not parameterizable.
- (Section 9.6) All the methods in this chapter make use of locating peaks in a transform space. It isn't sufficient to just identify local maxima. This section addresses how to find those peaks reliably.

At the end of this chapter, we also discuss how to find 3D shapes in range images (section 9.7) and correspondences in stereopsis (section 9.8) using parametric transform-based techniques.

9.2 The Hough Transform

First, let us prove an illustrative theorem.

Definition: Given a point in $\mathbb{R}^d$, and a parameterized expression defining a curve in that space, the parametric transform of that point is the curve that results from treating the point as a constant and the parameters as variables.

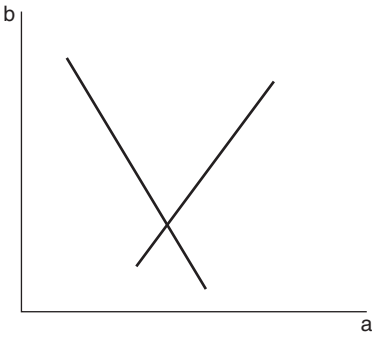


Figure 9.2 The intersection of two curves in the parameter space.

For example, the parametric transform of Eq. 9.1 produces

$$b = y - xa \tag{9.2}$$

which is itself a straight line in the 2-space $< a, b >$. Given the point $x = 3$, $y = 5$, then the parametric transform is the line in the $< a, b >$ space, $b = 5 - 3a$.

Theorem: If n points in a 2-space are colinear, all the parametric transforms corresponding to those points, using the form $b = y - xa$, intersect at a common point in the space $< a, b >$.

Proof. Suppose n points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ all satisfy the same equation

$$y = a_0 x + b_0. \tag{9.3}$$

Consider two of those points, (x_i, y_i) and (x_j, y_j). The parametric transforms of the points are the curves (which happen to be straight lines)

$$\begin{aligned} b &= y_i - x_i a \\ b &= y_j - x_j a. \end{aligned} \tag{9.4}$$

The intersection of those two curves in a, b is illustrated in Figure 9.2. Solving the two equations of Eq. 9.4 simultaneously results in

$$y_j - y_i = (x_j - x_i)a \tag{9.5}$$

and therefore $a = \frac{y_j - y_i}{x_j - x_i}$ as long as $x_j \neq x_i$. We substitute a into Eq. 9.4 to find b,

$$b = y_i - x_i \frac{y_j - y_i}{x_j - x_i}, \tag{9.6}$$

and we have the a and b values where the two curves intersect. However, we also know from Eq. 9.3 that all the x's and y's satisfy the same curve. By substituting Eq. 9.3 into Eq. 9.6, we obtain

$$b = (a_0 x_i + b_0) - x_i \frac{(a_0 x_j + b_0) - (a_0 x_i + b_0)}{x_j - x_i} \tag{9.7}$$

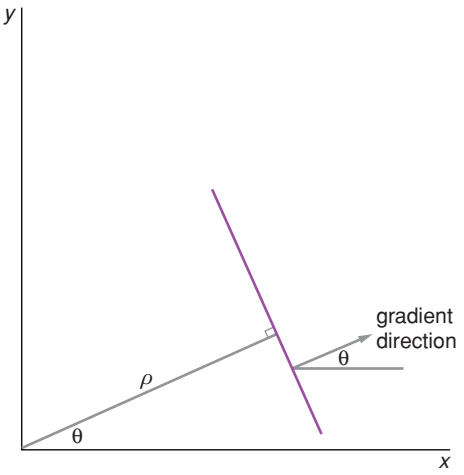


Figure 9.3 In the ρ, θ representation of a line, ρ is the perpendicular distance of the line from the origin, and θ is the angle made with the x axis. Note that if the line represents a brightness edge, at any point along the line, the direction of the gradient is normal to the line.

which simplifies to

$$b = (a_0 x_i + b_0) - x_i a_0 = b_0. \tag{9.8}$$

Similarly,

$$a = \frac{y_j - y_i}{x_j - x_i} = \frac{(a_0 x_j + b_0) - (a_0 x_i + b_0)}{x_j - x_i} = a_0. \tag{9.9}$$

Thus, for any two points along the straight line parameterized by a_0 and b_0, their parametric transforms intersect at the point $a = a_0$, and $b = b_0$. Since the transforms of any two points intersect at that one point, all such transforms intersect at that common point. QED.

Review of concept: Each POINT in the image produces a CURVE (possibly straight) in the parameter space. If the points all lie on a straight line in the image, the corresponding curves will intersect at a common point in the parameter space. Got that? Now, on to the next problem.

9.2.1 The Problem with Vertical Lines

What about vertical lines? Then the parameter a goes to infinity. This is not good. Perhaps we need a new form for the equation of a straight line. Here is a good one:

$$\rho = x \cos\theta + y \sin\theta. \tag{9.10}$$

Pick a value of ρ and θ, and then the set of points $\{(x_i, y_i)\}$ that satisfy Eq. 9.10 can be shown to be a straight line. There is a geometric interpretation of this equation that is illustrated in Figure 9.3.

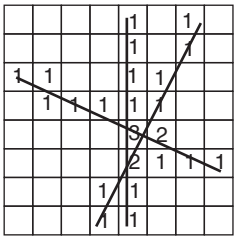


Figure 9.4 Three lines plotted through an accumulator array. The lines themselves are illustrated. Each accumulator is incremented whenever a line passes through that accumulator. Note that the vertical axis is ρ and the horizontal axis is θ.

This representation of a straight line has a number of advantages. Unlike the use of the slope, both of these parameters are bounded; ρ can be no larger than the largest diagonal of the image, and θ need be no larger than 2π. A line at any angle may be represented without singularity.

The use of this parameterization of a straight line solves one of the problems that confronts us, the possibility of infinite slopes. The other problem is the calculation of intersections.

9.2.2 How to Find Intersections – Accumulator Arrays

In the parameter space, it is not feasible to find all intersections of all curves and then to determine which of those are close together. Instead, we make use of the concept of an *accumulator array*. To create an accumulator array, we make an image, say 360 columns by 512 rows. We initialize each of the array elements to zero. It is often convenient to think of this array as an image, and from now on, we will refer to the elements of this special image as accumulators. Figure 9.4 illustrates plotting three straight lines through a very small accumulator array using the following algorithm:

For each point (x_i, y_i) in the edge image,

1. for all values of θ compute ρ.
2. at the point ρ, θ in the accumulator array, increase the value at that point by 1.

This algorithm results in multiple increments of those accumulators corresponding to intersections being increased more often. Thus the peaks in the accumulator array correspond to multiple intersections, and hence to proper parameter choices. Figure 9.5 illustrates an image with two straight edges and the corresponding Hough transform where the brightness of each pixel is the value of an accumulator. To understand why there are four intersections for two lines, consider that when one starts calculating ρ while varying θ, one will find points with negative ρ. Rather than testing negative values and reflecting, it is faster to simply allow ρ to be negative; which makes the accumulator twice as big. With the larger accumulator, including $0 \rightarrow 350^{o}$ and both positive and negative ρ, there is a redundancy. For example, in Figure 9.5, the leftmost peak is at 66^{o} and the third peak is

Figure 9.5 On the left is an image with two line segments that have distinctly different slopes and intercepts, but whose actual positions are corrupted significantly by noise. On the right is the corresponding Hough transform. On the transform, the vertical axis is ρ, and θ is the horizontal axis.

at 247°, a difference of almost exactly 180°. These two peaks actually correspond to the same point.

9.2.3 Reducing Computational Complexity Using Gradient

The computational complexity of parametric transforms can be quite high. The Hough transform is sometimes referred to as the "coffee transform," because for many years one could command a computer to perform a Hough transform, and then drink an entire cup of coffee before it finished. For example, suppose we are performing a Hough transform in which the size of the image is 512×512, and we wish an angular resolution of one degree. Our accumulator array is thus $(512 \times 2\sqrt{2}) \times 360$ (since ρ can be positive or negative). The iteration consists of calculating ρ for each value of θ, so we calculate and increase 360 accumulators for each edge pixel in the image.

One way to reduce the computation is to observe that the edge points in the image are often the output of some gradient operator, and if we have both the magnitude and direction of that gradient, we can make use of that information to reduce the computational complexity. To see this, refer to Figure 9.3. If we know the gradient at a point, we know the direction of the edge, and we therefore know θ. Thus, we need only to compute one value of θ, instead of 360 of them, and we need to increment only one accumulator – a 360:1 speed-up. Not bad!

Of course (you should be used to caveats by now), there are some problems with this method. The first is that the direction returned by most gradient operators is not particularly accurate. (Go back and look at section 5.8.1). Therefore, the one cell that you are incrementing may not be precisely located. There is an easy partial solution to that: the trick is to not increase a point, but to increase a neighborhood. For example, you might increase the point calculated by two, and increment the neighborhood of that point by one. (Yes, that is a Gaussian of sorts).

You can also use the magnitude of the gradient in the process of incrementing the accumulator array. In the description presented earlier, we suggested that you threshold the gradient image, and increment the accumulator array by 1 at the points corresponding to the edge points in the image. Another approach is to increment by the magnitude of the gradient, or a value proportional to the magnitude. Of course, this requires that you use a

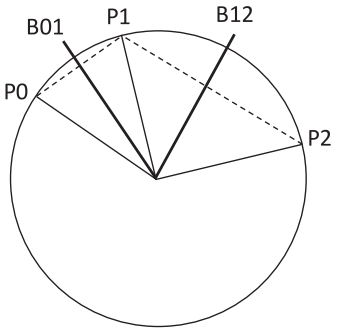


Figure 9.6 The circle containing points P0, P1, and P2. The two chords are shown as dashed lines, and the perpendicular bisectors of the chords are B01 and B12.

floating point representation for the accumulator array, but that should be no problem if a floating point accumulator is used.

In summary, you can trade off computation in image space (computation of magnitude and direction of the gradient) for computation in parameter space, and this trade-off can result in substantial speed-ups. You will see this again later in this chapter. Other heuristics [9.12] can also result in speed-ups.

9.3 Finding Circles

A generalization of the Hough transform can be used to encompass detection of segments of circles. The practicality of the technique is ensured by use of low dimensionality accumulator arrays.

9.3.1 Derivation of the Location of a Circle Represented by Any Three Noncollinear Pixels

Given any three noncollinear points in the plane, this subsection will review how the center and radius of a circle passing through those three points can be found. In section 9.3.2 this result will be integrated into a parametric transform that will identify the circle that passes through these three points.

In Figure 9.6, three points are shown, P0, P1, and P2. The circle that passes through all three is shown. The chords P0P1 and P1P2 are shown. The perpendicular bisectors of those chords are labeled as B01 and B12. The intersection of B01 and B12 is the center of the circle, and the radius is the distance from that point of intersection to any of the three points, P0, P1, and P2. In the case where the bisectors B01 and B12 are parallel, they will not have a finite intersection. Any three noncollinear points will lie on a circle with the center found by intersection of the perpendicular bisectors of line segments joining the points and the radius equal to the distance from the center to one of the points. The circle containing P0, P1, and P2 is defined by the following equation in the x-y plane.

$$(x-h)^2 + (y-k)^2 = R^2 \tag{9.11}$$

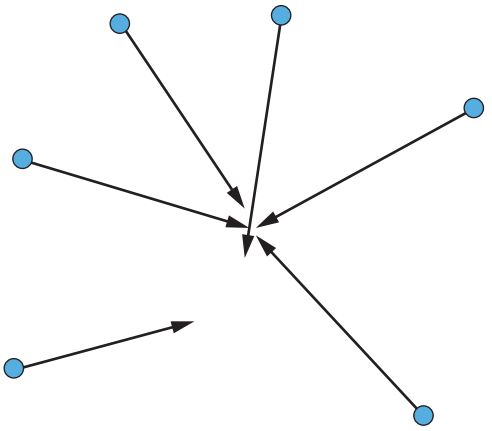


Figure 9.7 All the extensions of the gradient vectors tend to intersect at a point.

where $C = (h, k)$ is the center of this circle with radius R. But how do we use the techniques of parametric transformations to find the parameters, given that we know some information about the circle? Or, what if we know nothing about the circle? Those questions are addressed in the next subsection.

9.3.2 Finding Circles When the Origin Is Unknown but the Radius Is Known

Equation 9.11 describes an expression in which x and y are assumed to be variables, and h, k, and R are assumed to be parameters. As before, let us rewrite this equation interchanging the roles of parameters and variables.

$$(h - x_i)^2 + (k - y_i)^2 = R^2. \tag{9.12}$$

In the space (h, k) what geometric shape does this describe? You guessed it, a circle. Each point in image space (x_i, y_i) produces a curve in parameter space, and if all those points in image space belong to the same circle, where do the curves in parameter space intersect? You should be able to figure that one out by now. Now, what if R is also unknown? It is the same problem. However, instead of allowing h to range over all possible values and computing k, we must now allow both h and k to range over all values and compute R. We now have a three-dimensional parameter space. Allowing two variables to vary and computing the third defines a surface in this 3-space. What type of surface is this (an ellipse, a hyperboloid, a cone, a paraboloid, a plane)?

9.3.3 Using Gradient Information to Reduce Computation in Finding Circles

Suppose we know we have only one circle, or a portion of a circle in the image. How can we find the center with a minimum of computation? One approach is as follows: At every edge point, compute the gradient vector. The accumulator is isomorphic to the image. At every edge point, walk along the gradient vector, incrementing the accumulator as you go. Just as before, the maximum of the accumulator will indicate the center of the circle. If the interior of the circle is brighter than the outside, then all the gradient vectors will point to

approximately the same point as illustrated in Figure 9.7. If the inside is darker, then the gradient vectors will all point away, but in either case, the extensions of the vectors will all intersect at a common point.

What if you know the radius? If you know the direction (from the gradient), and you know the distance (from knowing the radius), then you know the location of the center of the circle (at least to within an inside/outside ambiguity). Let's say you have a point at (x_i, y_i) that you believe might be on a circle of radius R, and assume the gradient at that point has a magnitude of M and a direction of θ. Then, the location of the center of the circle is at

$$x_0 = x_i - R\cos\theta \quad y_0 = y_i - R\sin\theta. \tag{9.13}$$

Again, the accumulator array should be incremented over a neighborhood rather than just a single accumulator, and it will prove efficacious to increment by an amount proportional to M.

9.4 Finding Ellipses

Only if the major axis of the ellipse is horizontal does an ellipse satisfy the equation

$$\frac{x^2}{\alpha^2} + \frac{y^2}{\beta^2} = 1, \quad \alpha > \beta \tag{9.14}$$

where α and β are scalar constants. A similar equation results if the major axis is vertical, but for all other rotations of the ellipse relative to the coordinate axes, only an equation of the form

$$ax^2 + cy^2 + bxy + dx + ey + f = 0 \tag{9.15}$$

is sufficient. But this is the equation of a general conic, and is not constrained to be an ellipse. One could add the constraint, $b^2 - 4ac < 0$, but adding this constraint would convert the fitting process into a much more difficult problem. Instead, Xie and Ji [9.11] provide a simple algorithm for finding ellipses that uses an accumulator-based philosophy.

The algorithm proceeds by choosing two points to be the major axis of an ellipse, and then looking for consistency to determine that this pair is in fact the major axis.

Figure 9.8 illustrates the geometric features of an ellipse. Examination of that figure reveals that knowledge of the major axis and its end points also determines the center and the orientation, leaving only the length of the minor axis to be determined. The following algorithm outlines the method:

1. Set parameters Mm and MM, the minimum and maximum major axis length, as well as mm and mM, the minimum and maximum minor axis length.
2. Scan the image finding points of strong gradient, and store each of those points in an array P containing, x, y, m, θ, where m is the magnitude and θ the direction of the gradient. This array could be rather large.

In addition, allocate space to hold a one-dimensional accumulator of lengths of the minor axis.

3. Scan through P and choose a point $\mathbf{x}_1$ that could be the end of the major axis of some ellipse. It could not be the end, for example, if the direction of the gradient would force the major axis to be off the image.
4. Now, choose a candidate for the other end of the major axis, $\mathbf{x}_2$. Various heuristics may be used to reduce the search for $\mathbf{x}_2$. For example 1) the other end must satisfy $Mm \leq |\mathbf{x}_1 - \mathbf{x}_2| \leq MM$, or 2) knowledge of the gradient direction at $\mathbf{x}_1$ provides information of where to search for $\mathbf{x}_2$.
5. The half length of the major axis, a, is $|\mathbf{x}_1 - \mathbf{x}_2|/2$, the origin, $\mathbf{x}_0$, is at $(\mathbf{x}_1 + \mathbf{x}_2)/2$, and the orientation of the ellipse is $\theta = tan^{-1}(y_2 - y_1)/(x_2 - x_1)$.
6. Choose a point $\mathbf{x}$ such that $|\mathbf{x} - \mathbf{x}_0| < a$. Let $d = |\mathbf{x} - \mathbf{x}_0|$, and let $f = |\mathbf{x}_2 - \mathbf{x}|$.
   (a.) Consider the triangle, $\mathbf{x}$, $\mathbf{x}_0$, and $\mathbf{x}_2$, illustrated in Figure 9.8, which has an internal angle of τ. Since $a^2 + d^2 - f^2 = 2ad\cos\tau$, we can calculate $\cos\tau$:

$$\cos\tau = \frac{a^2 + d^2 - f^2}{2ad}.$$

   Let $2b$ denote the length of the minor axis, so that b is the "half length" of that axis. Then

$$b^2 = \frac{a^2 d^2 \sin^2\tau}{2ad}.$$

   (b.) Increment the accumulator for the length b.
   (c.) If any more pixels exist that satisfy the conditions of step 6, then repeat step 6 with those pixels.
7. Find the maximum of the accumulator. If the maximum is sufficiently high, then $\mathbf{x}_1$ and $\mathbf{x}_2$ are the endpoints of the major axis of an ellipse. Save this information. If the maximum of the accumulator is too small, continue scanning with step 3.
8. Clear the accumulator
9. Remove all the points just found on this ellipse from P.
10. Continue until the end of P is reached.

This algorithm is fast, efficient, and relatively easy to program and to understand. It has only one problem: if there is a real ellipse, but one end or the other is occluded, the algorithm will not work.

9.5 The Generalized Hough Transform

So far, we have assumed that the shape we are seeking can be represented by an analytic function, representable by a set of parameters. The concepts we have been using, of allowing data components that agree to "vote," can be extended to generalized shapes. Now, let the region be arbitrarily shaped, and suppose the orientation, shape, and scale are known. Our first problem is to figure out how to represent this object in a manner suitable for use by Hough-like methods. Following is one such approach [9.1]:

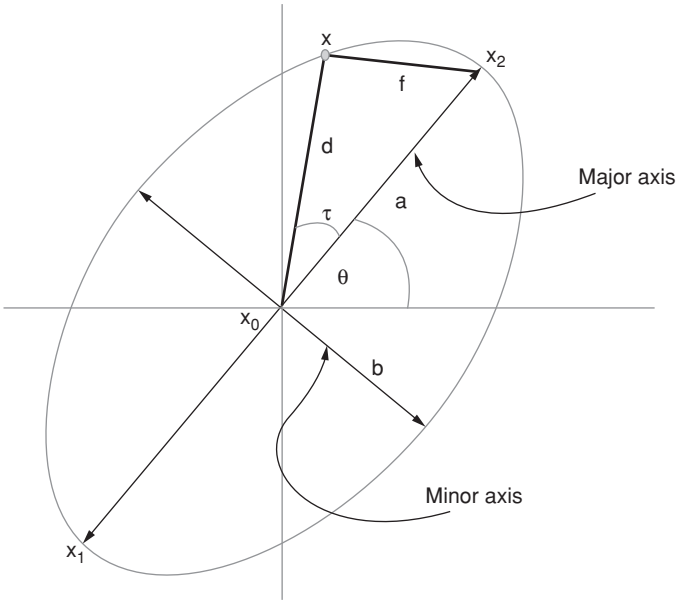


Figure 9.8 If one knows the major axis and the minor axis, then one knows the ellipse.

First, define some reference point. The choice of a reference point is arbitrary, but the center of gravity is convenient. Call that point O. For each point P_i on the boundary, calculate both the gradient vector at that point and the vector OP_i from the boundary point to the reference. Quantize the gradient direction into, say, n values, and create a table R with n rows. Each time a point P_i on the boundary has a gradient direction with value θ_i ($i = 1, \ldots, n$), a new column is filled in on row i, using $T[i, k] = OP_i$, where k is the index of the entry on row i. Thus, the fact that multiple points on the boundary may have identical gradient directions is accommodated by placing a separate column in the table for each entry.

In Figure 9.9 the shape and three entries in the R-table are shown.

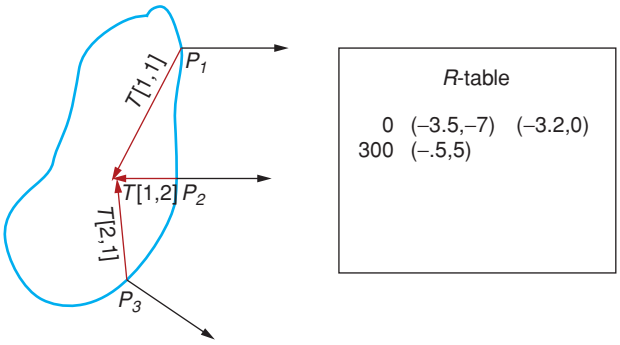


Figure 9.9 A simple figure with three points, and the resulting R-table. $T[i, k]$ is the kth vector on the ith row of the R-table. The symbols in the figure and the numbers in the R-table may be resolved by: $T[1, 1] = (-3.5, -7)$, $T[1, 2] = (-3.2, 0)$, and $T[2, 1] = (-.5, 5)$.

To utilize such a shape representation to perform shape matching and location, we use the algorithm below.

Algorithm for the Generalized Hough Transform

1. Form a 2D accumulator array, which will be used to hold candidate image locations of the reference point. Initialize the accumulator to zero.
2. For each edge point, P_i
   (a) Compute the gradient direction, and determine which row of the R-table corresponds to that direction. Let that row be row i.
   (b) For each entry, k, on row i,
      i. Compute the location of the candidate center by adding the stored vector to the boundary point location: $\mathbf{a} = T[i, k] + P_i$.
      ii. Increment the accumulator at $\mathbf{a}$.

In the assignments for this chapter, issues of invariance are discussed.

9.6 Finding the Peaks

The peaks of the accumulator array are, ideally, at or near the true parameter values, but in general they may be displaced by noise in the original data. There are several ways to approach this problem, including some sophisticated methods [9.2, 9.9] and some simple ones.

In the simple problem where there should be a single peak, it is often sufficient to look for a single maximum. However, as we learned in Chapter 5, kernel operators don't estimate directions very accurately. Furthermore, as a consequence of sampling and noise, image points aren't precisely where we think they are. Therefore, the lines drawn in the accumulator do not intersect precisely and make perfect, sharp peaks. Such peaks tend to be "bumpy," a single peak may be multimodal. Some simple blur is usually sufficient to smooth the bumps and produce a single local maximum. However, smoothing is not always sufficient. Figure 9.10 illustrates a close-up of a typical accumulator "peak," that is seen here to not be a single peak at all.

A strategy for the problem where more than one significant peak could occur is as follows:

1. Search the accumulator for local maxima over a threshold in height.
2. If two of these maxima are "close" to each other, combine them.
3. Go to step 1 until no significant new peaks are found.

The general multi-peak problem is fundamentally a clustering problem [9.2], so any algorithm that does clustering well is also good at peak finding.

As was discussed in Chapter 8, clustering is the process of finding natural groupings in data. The obvious application is to find the best estimate of the peak(s) of the accumulator, however, these ideas are much broader in applicability than just finding the mode of a distribution. Figure 9.11 illustrates the use of k-means clustering to find peaks in the

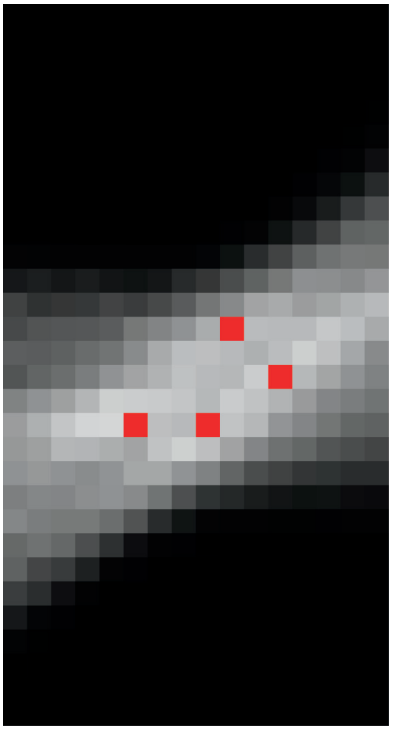


Figure 9.10 The region considered to be a "peak" in a Hough accumulator. This bright area covers from 56 to 78 degrees on the horizontal axis of the accumulator and 200 to 208 units of distance on the vertical axis. Four separate local maxima are identified by red dots.

accumulator. To find n peaks, just start clustering with n distinct cluster centers, and let the cluster centers move. Clustering has been discussed in detail in section 8.3.

9.7 Finding 3D Shapes – The Gauss Map

The "Gauss map" provides an effective way to represent operations in range images. It is philosophically a type of parametric transform. The concept is quite simple: First, tessellate

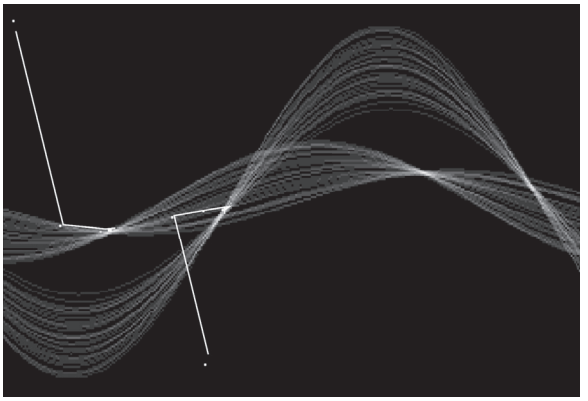


Figure 9.11 Movement of two cluster centers from random starting points to peaks.

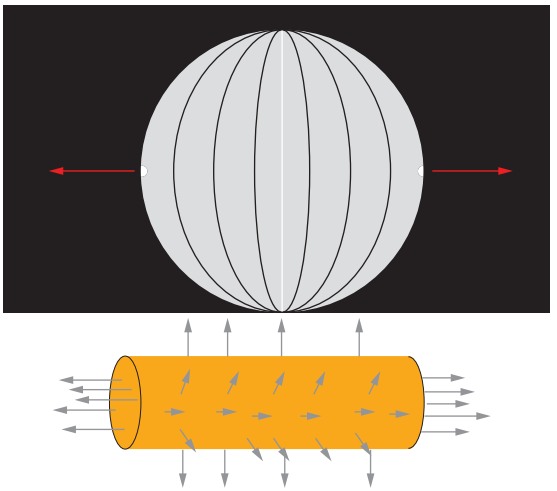


Figure 9.12 A shape, a cylinder with two flat ends, and the Gauss map representation for that shape. Normal vectors on the cylinder are illustrated by arrows. The cylinder is rotated slightly toward the viewer so the viewer can see one end. This makes the arrows vary in apparent length. The Gauss sphere itself is illustrated on a black background, and the areas that have been incremented are indicated as white. Two large red arrows have been drawn beside the sphere to indicate the normal directions at the two peaks.

the surface of a sphere, into whatever size patches you wish. This tessellation will determine the resolution of the map. Associate a counter (accumulator) with each cell of the tessellation. Now consider a range image. In the range image, calculate the surface normal at each pixel, and increment the accumulator of the Gauss map that has the same surface normal. This provides essentially a histogram of normal vectors, which in turn can be used to recognize the orientation of the object in the range image.

Determining the orientation of a surface from the Gauss map is accomplished by first identifying significant peaks in the map. The location of the peak on the map corresponds to the orientation of the surface, and the fact that the peak is large indicates that there are many surface patches with that orientation on the surface.

Figure 9.12 illustrates a 3D shape, a cylinder, and its Gauss sphere representation.

9.8 Finding the Correspondences – Parametric Consistency in Stereopsis

We will now briefly address the problem of stereopsis and provide an improvement based on accumulator arrays.

Although we won't see the details until Chapter 12, you know already that when a person observes the same object separately with each eye, the object seems to move (horizontally) between the two views. This *disparity* is the distance in pixels between two corresponding pixels that can be calculated as

$$d = \frac{BF}{z} \tag{9.16}$$

where z is the distance to the point corresponding to those two pixels, B is the baseline (the distance between the two cameras/eyes), and F is the focal distance of either of the

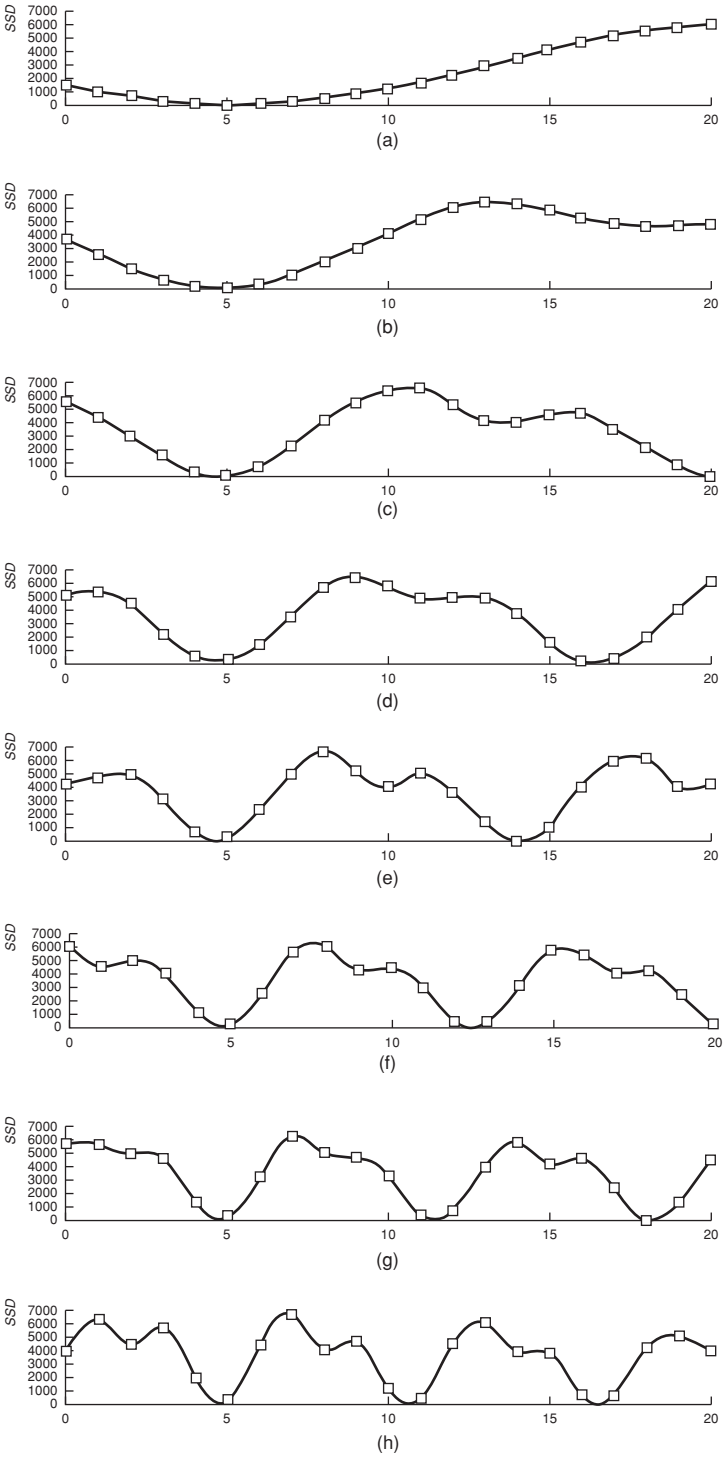


Figure 9.13 The result of matching a template extracted from one image in a stereo pair with a second and third image, as a function of the location of the inverse distance to the matching point. Figure from [9.8]. Used with permission.

cameras (All the cameras are assumed to be identical.) The most difficult problem is the determination of which pixels in each camera image correspond to the same points in 3-space. Suppose a small window is extracted from the leftmost image and use the sum of squared differences (SSD) to template match that small window along a horizontal line in the other image. One could graph that objective function vs. the disparity, or the inverse distance ($1/z$). It is convenient to use the inverse distance and we will typically find that the match function has multiple minima, as illustrated in Figure 9.13. If an image is taken from a third or fourth camera, with a different baseline from camera 1, similar nonconvex curves are found. However, all the curves will have minima at the same point, the correct disparity. Immediately, we have a consistency! A new function (an accumulator) can be formed, the sum of such curves taken from multiple baseline pairs. We find that this new function (called the SSD-in-inverse-distance) has a sharp minimum at the correct answer. Okutomi and Kanade [9.8] have proven that this function always exhibits a clear minimum at the correct matching position, and that the uncertainty of the measurement decreases as the number of baseline pairs increases.

9.9 Conclusion

In this chapter, the concept of parametric transforms was introduced in the form of the Hough transform and its descendents, all of which use accumulator arrays. An accumulator array allows for easy detection of consistency, since "things" that are consistent all add to the same accumulator cell, or at least to nearby accumulator cells. By constructing the accumulator array by adding hypotheses, we also gain a measure of noise immunity, since inconsistent solutions tend not to contribute to the globally consistent solution.

The term "Hough transform" is really a misnomer when applied to finding circles, ellipses, etc. Only the original work by Hough [9.4] to find straight lines should really be referred to by the term. However, many authors have come to use the term "Hough transform" when they really mean "parametric transform." The exception to this is the Generalized Hough Transform [9.1], which has no parameters except the properties of the curve itself.

Since curvature represents rates of change of normal direction, the Gauss map can be related back to curvature. The invertibility of the Gauss map and invariance under rotation and translation are discussed in more detail in [9.5]. The map also finds application in identifying the vanishing points in an image [9.6].

Clustering, mentioned in this chapter as a way to find peaks in the accumulator and in section 8.3 as a way to group colors, is a general and powerful tool. For example, McLean and Kotturi [9.7] use one version of clustering [9.3] to locate the vanishing points in an image.

The topics listed in this chapter do not include all applications of accumulator methods. However the literature provides a variety of examples such as finding parabolas [9.10].

9.10 Assignments

Assignment 9.1: In the directory named "`leadhole`" is a set of images of wires coming through circuit board holes. The holes are roughly circular and black. Use parametric

transform methods to find the centers of the holes. This is a project and will require a formal report. In your report, show the image, zoomed to a viewable size. Mark the pixels that are elements of the boundary of the hole, and the center of the hole. This marking process can be easily accomplished by simply setting those pixels to very bright. Process as many images as possible. If your method fails in some cases, discuss why.

Assignment 9.2: You are to use the Generalized Hough Transform approach to both represent an object and to search for that object in an image. (The object is a perfect square, black inside on a white background, centered at the origin, with sides two units long.) You only have five points to use, those at (0, 1), (1, 0), (1, 0.5), (−1, 0), and (0, −1). Fill out the "R-table" that will be used in the generalized Hough transform of this object.

The example table below contains four rows. You are not required to fill them all in, and if you need more rows, you can add them.

p1(x,y)	p2(x,y)

Assignment 9.3: Let P1 = [x1, y1] = [3, 0] and P2 = [x2, y2] = [2.39, 1.42] be two points, that lie approximately on the boundary of the same circular disc. We do not know *a priori* whether the disk is dark inside or bright. The image gradients at P1 and P2 are $5\angle 0$ and $4.5\angle \pi/4$ (using polar notation).

Use Hough methods to estimate the location of the center of the disc, and radius of the disc, and determine whether the disc is darker or brighter than the background.

Assignment 9.4: Let P1 = [x1, y1] = [3, 0], P2 = [x2, y2] = [2.39, 1.42,] and P3 = [1.1, 1.99] be three points that lie approximately on the boundary of the same disc. We do not know *a priori* whether the disk is dark inside or bright. The image gradients at P1, P2, and P3 are $5\angle 0$, $4.5\angle \pi/4$, and $4.9\angle \pi/2$ (using polar notation), respectively.

Use Hough methods to estimate the location of the center of the disc, and the radius of the disc, and determine whether the disc is darker or brighter than the background.

Assignment 9.5: Noise is always present, even though parametric transforms like the Hough do a remarkably good job of reducing it. Could you apply the concepts we learned in Chapter 6 such as VCD, to an accumulator to reduce the noise in that accumulator? Discuss.

Assignment 9.6: Is the Generalized Hough Transform (GHT) as described in section 9.5 invariant to zoom? That is, suppose you have two regions, A and B, which are identical except for the fact that B is a zoomed version of A. Will the GHT described in that section conclude that they are the same shape? If not, how would you modify that algorithm to make it invariant to zoom?

Assignment 9.7: In Figure 9.11 a Hough accumulator is illustrated. Invent a way to find the peaks using a clustering algorithm. (Do *not* simply find the brightest point).

Suggestions might include weighting each point by the square of the accumulator value, doing something with the exponential of the square of the value, etc.

Bibliography

[9.1] D. Ballard. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2), 1981.

[9.2] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(8), 1995.

[9.3] T. Hofmann and J. Buhmann. Pairwise data clustering by deterministic annealing. *IEEE Trans. Pattern Anal. and Machine Intel.*, 19(1), 1997.

[9.4] P. V. C. Hough. Method and means for recognizing complex patterns. *U.S. Patent*, 3069654, 1962.

[9.5] P. Liang and C. Taubes. Orientation-based differential geometric representations for computer vision applications. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(3), 1994.

[9.6] E. Lutton, H. Maitre, and J. Lopez-Krahe. Contribution to the determination of vanishing points using the Hough transform. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(4).

[9.7] G. McLean and D. Kotturi. Vanishing point detection by line clustering. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(11), 1995.

[9.8] M. Okutomi and T. Kanade. A multiple-baseline stereo. *IEEE Trans. Pattern Anal. and Machine Intel.*, 15(4), 1993.

[9.9] J. Princen, J. Illingworth, and J. Kittler. Hypothesis testing: A framework for analyzing and optimizing Hough transform performance. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(4), 1994.

[9.10] H. Wechsler and J. Sklansky. Finding the rib cage in chest radiographs. *Pattern Recognition*, 9, 1977.

[9.11] Y. Xie and Q. Ji. A new efficient ellipse detection method. In *ICRA*, 2002.

[9.12] Ylä-Jääski and N. Kiryati. Adaptive termination of voting in the probabilistic circular Hough transform. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(9), 1994.

10 Representing and Matching Shape

Shape is what is left when the effects associated with translation, scaling, and rotation are filtered away.

– David Kendall

10.1 Introduction

In this chapter, we assume a successful segmentation, and explore the question of characterization of the resulting regions. We begin by considering two-dimensional regions that are denoted by each pixel in the region having value 1 and all background pixels having value 0. We assume only one region is processed at a time, since in studying segmentation, we learned how to realize these assumptions.

When thinking about shapes, and measures for shapes, it is important to keep in mind that certain measures may have *invariance* properties. That is, a measure may remain the same if the object is (for example) rotated. Consider the height of a person in a picture – if the camera rotates, the apparent height of the person will change, unless, of course, the person rotates with the camera.

The common transformations considered in this chapter are those described by some linear operation on the shape matrix, discussed in section 4.2.3.

In the remainder of this chapter, we will be constantly thinking about operations that exhibit various invariances, and can be used to match shapes of regions.

- (Section 10.2) To understand invariances, we must first understand the deformations that may alter the shape of a region, and most of those can be described by matrix operations.
- (Section 10.3) One particularly important matrix is the covariance matrix of the distribution of points in a region, since the eigenvalues and eigenvectors of that matrix describe shape in a very robust way.
- (Section 10.4) In this section, we introduce some important features used to describe a region. We start from some simple properties of a region like perimeter, diameter, and thinness. We then extend the discussion to some invariant features (to various linear transformations) like moments, chain codes, Fourier descriptors, and the medial axis.
- (Section 10.5) Since, in earlier sections, we have represented the regions by sets of numbers called *features*, in this section, we discuss how to match such sets of numbers.

- (Section 10.6) From the boundary of a homogeneous region, the shape is determined. In this section, four different approaches to describing and matching boundaries are discussed. These descriptors represent more general methods for describing shapes at a more abstract level.
- (Section 10.7) A boundary could be written as a long vector, and so may be thought of as a point in a very high dimensional space. This provides yet another way to match and to interpolate shapes. In this section, the concept is introduced and demonstrated as a method to interpolate between shapes.

10.2 Linear Transformations

One important topic in this chapter is invariance to various types of linear transformations on regions. That is, consider all the pixels on the boundary as we did in creating the shape matrix, and write the x, y coordinates of each pixel as a 2-vector, and then operate on that set of vectors. In general, no distinction needs to be made (here) between whether we are thinking of the entire region or just its boundary.

10.2.1 Rigid Body Motions

The first transformations of interest are the orthogonal transformations such as $R_Z = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$, which operate on the coordinates of the pixels in a region to produce new coordinate pairs. For example, $\begin{bmatrix} x' \\ y' \end{bmatrix} = R_Z \begin{bmatrix} x \\ y \end{bmatrix}$ where R_Z is as defined above, represents a rotation about the Z axis, defined to be the axis normal to the image plane. Given a region s, we can easily construct its shape matrix, which we denote by S, in which each column contains the x, y coordinates of a pixel in the region. For example, suppose the region $s = \{(1, 2), (3, 4), (1, 3), (2, 3)\}$, then the corresponding coordinate matrix S is $S = \begin{bmatrix} 1 & 3 & 1 & 2 \\ 2 & 4 & 3 & 3 \end{bmatrix}$. We can apply an orthogonal transformation such as a rotation, to the entire region by matrix multiplication

$$S' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 3 & 1 & 2 \\ 2 & 4 & 3 & 3 \end{bmatrix}.$$

That works wonderfully well for rotations, but how can we include translation in this formalism?

The first way to add translation is to rotate first, using a 2×2 rotation and then add the translation, that is

$$S' = R_z S + \begin{bmatrix} dx \\ dy \end{bmatrix}, \tag{10.1}$$

and this works fine, but it requires two operations, a matrix multiplication and an addition. Wouldn't it be nice if it could be done in a single matrix multiplication?

This can be accomplished by augmenting the rotation matrices by adding a row and a column, all zeros except for a 1 in the lower right corner. With this new definition, a rotation has this form:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The definition of a point is also augmented by adding a 1 in the third position, so the coordinate pair (x, y) in this new notation becomes $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$.

Now, translation and rotation can be combined into a single matrix representation (called a *homogeneous transformation matrix*). This is accomplished by changing the third column to include the translation. For example, to rotate a point about the origin by an amount θ, and then translate it by an amount dx in the x direction and dy in the y direction, we perform the matrix multiplication

$$\mathbf{x}' = \begin{bmatrix} \cos\theta & -\sin\theta & dx \\ \sin\theta & \cos\theta & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{10.2}$$

Thus, it is possible to represent rotation in the viewing plane (about the z axis) and translation in that plane by a single matrix multiplication. All the transformations mentioned above are elements of a class of transformations called *isometries*, which are characterized by the fact that they may move an object around, but they do not change its shape or its size. These are sometimes also referred to as *rigid body motions*.

10.2.2 Affine Transformations

If we generalize the idea of a rigid body motion just a bit, to allow the object to also become larger, we get *similarity transformations*.[1] A similarity transform allows scale change (e.g., zoom).

But what can we do, if anything, to represent rotations out of the camera plane? To answer this, we need to define an *affine transformation*. An affine transformation of the 2-vector $\mathbf{x} = [x, y]^{\mathrm{T}}$ produces the 2-vector $\mathbf{x}' = [x', y']^{\mathrm{T}}$ where

$$\mathbf{x}' = A\mathbf{x} + \mathbf{b}, \tag{10.3}$$

and $\mathbf{b}$ is likewise a 2-vector. This looks just like the similarity transformations mentioned above, except that we do not require the matrix A be orthonormal, only nonsingular. An affine transformation may distort the shape of a region. For example, shear may result from an affine transformation, as illustrated in Figure 10.1.

As you probably recognize, rotation of a planar object out of the field of view is equivalent to an affine transformation of that object. This gives us a (very limited) way to think about rotation out of the field of view. If an object is nearly planar, and the rotation out of the field of view is small, that is, nothing gets occluded, the projection of this 3D motion onto the viewing plane can be approximated as a 2D affine transformation. For example,

[1] Yes, this is from the same root as similar triangles.

Figure 10.1 Affine transformations can scale the coordinate axes. If the axes are scaled differently, the image undergoes a shear distortion. The "blob" image mentioned in Chapter 8 is shown here after rotation by 0.4 rad and with shears of .7 in the column direction and 1.5 in the row direction.

Figure 10.2 shows some images of an airplane that are all nearly affine transformations of each other. The matrix that implements an affine transform (after correction for translation) can be decomposed into its constituents, rotation, shear, and scale changes (respectively):

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix} \begin{bmatrix} 1 & \delta \\ 0 & 1 \end{bmatrix}. \tag{10.4}$$

As before, we can use homogeneous coordinates to combine affine deformation and translation using

$$\begin{bmatrix} \mathbf{y} \\ 1 \end{bmatrix} = \begin{bmatrix} A & \mathbf{b} \\ 0,0,0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}. \tag{10.5}$$

Now, what does one do with these concepts of transformations? One can correct transformations and align objects together through inverse transformations to assist in shape analysis. For example, one can correct for translation by shifting so the centroid is at the origin. Correcting for rotation involves rotating the image until the principal axes of the object are in alignment with the coordinates axes.

Finding the principal axes is accomplished by a linear transformation that transforms the covariance matrix of the object (or its boundary) into the unit matrix. This transformation is related to the Whitening transformation and the K-L transform. Unfortunately, once such a

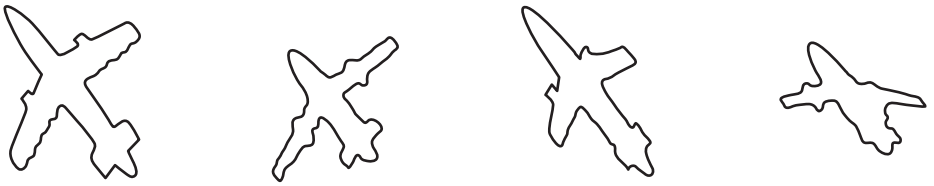


Figure 10.2 Aircraft images that are affine transformations of each other. (From [10.3]). Used with permission.

transformation has been done, Euclidean distances between points in the region have been changed [10.41].

10.2.3 Norms and Metrics

In the previous paragraph, the word "distance" occurs. Although one usually thinks that "distance" means "Euclidean distance," in this book, this word is used many times and in many forms. For that reason, we should be just a bit more rigorous about exactly what the concept means. The Euclidean distance (for example) is a type of measure called a "metric."

Any metric $m(a, b)$ in a d-dimensional space has the following properties: $m : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$

$$\begin{aligned} \forall_{a,b} m(a,b) \geq 0 \quad & \text{nonnegativity} \\ \forall_{a} m(a,a) = 0 \quad & \text{identity} \\ \forall_{a,b} m(a,b) = m(b,a) \quad & \text{symmetry} \\ \forall_{a,b,c} m(a,b) + m(b,c) \geq m(a,c) \quad & \text{triangle inequality} \end{aligned} \tag{10.6}$$

We will have several opportunities to examine metrics in later chapters.

Frequently, we will also find ourselves using the term *norm*, usually in the context of the norm of a vector or of a matrix. Usually, the norm operation is denoted by the presence of two vertical bars on each side of the vector, as in $||\mathbf{x}||$. However, you may encounter the notation of a single vertical bar, (as in absolute value) being used as a norm. If $|| \cdot ||$ is a scalar measure of a vector, it is a norm if it has the following properties, for any vectors $\mathbf{x}$ and $\mathbf{y}$ and any scalar α:

$$\begin{aligned} &\text{if } \mathbf{x} \neq 0 \text{ then } ||\mathbf{x}|| > 0, \\ &||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}||. \end{aligned}$$

We frequently use the term *p-norm*, where p is some integer. The *p-norm* of a vector $\mathbf{x}$ is defined by the sum $(\sum_i |x_i|^p)^{\frac{1}{p}}$. Here are some common examples:

L_0 **norm** This is simply the number of elements in the vector.

L_1 **norm** $\sum_i |x_i|$. Here x_i is the i^{th} element of the vector, and the vertical bar is the absolute value of the element. If the vector is resulted from the difference of two vectors, this norm is called the *city block distance* because it is the distance you would walk if you could not cut across a block. It is also (for the same reason) sometimes called the *Manhattan distance*. Figure 10.3 illustrates an application of the Manhattan distance.

L_2 **norm** $\sqrt{\sum_i |x_i|^2}$. This is the usual definition of the magnitude of a vector, and is equal to $\sqrt{\mathbf{x}^{\mathrm{T}}\mathbf{x}}$. If the vector resulted from the difference of two vectors, this norm is called the *Euclidean distance*.

L_∞ **norm** $(\sum_i |x_i|^\infty)^{\frac{1}{\infty}}$. Obviously, you can't take something to the infinity power, but as p gets larger, $|x_i|^p$ becomes dominated by the largest element. Therefore, the L-infinity norm is simply the maximum.

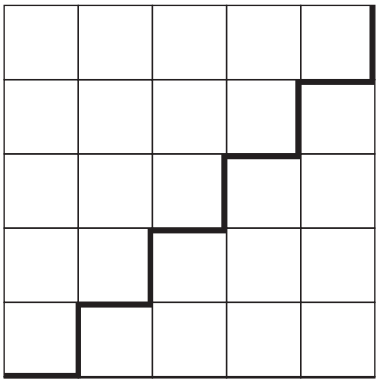


Figure 10.3 The Manhattan distance from one corner of a 5×5 square to the opposite is (surprisingly) 10, not $5\sqrt{2}$.

As an example to encourage thinking about norms, consider the set of positive real numbers, denoted $\mathbb{R}_{\geq 0}$, and think about vectors constructed of positive real numbers. Now, define an operator that might be a norm by:

$$L_s(\mathbf{a}) = \frac{1}{n}\sum_{i=1}^{n} a_i \tag{10.7}$$

Consider $L_s(\mathbf{a})$. Certainly this can only be zero if $\mathbf{a}$ is the zero vector. Second, consider $L_s(\mathbf{a} + \mathbf{b})$. Since

$$\frac{1}{n}\sum_{i=1}^{n} a_i + \frac{1}{n}\sum_{i=1}^{n} b_i = \frac{1}{n}\sum_{i=1}^{n}(a_i + b_i), \tag{10.8}$$

we see that this operator meets the two requirements stated above, so it appears to be a norm. At least when applied to vectors in $\mathbb{R}_{\geq 0}$. In fact, it is almost the L_1 norm.

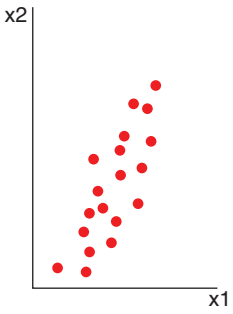
But, is this really a norm or is It just the average of the elements of the vector after all? Is there something in the previous paragraph that is tricky?

Well, yes, sort of. The formal definition of a norm given above assumes the set of vectors to which it applies is a vector space, and $\mathbb{R}_{\geq 0}$ is not a vector space (unless we are willing to change that definition too) because it does not contain additive inverses.

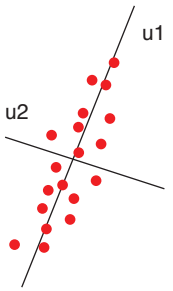
Still, L_s might be useful. It is certainly simpler than $\sqrt{r \times r + g \times g + b \times b}$ when finding a grayscale representation for a color pixel.

10.3 The Covariance Matrix

Consider the distribution of points illustrated in Figure 10.4a. Each point may be exactly characterized by its location, the ordered pair, (x_1, x_2), but neither x_1 nor x_2, by itself, is adequate to describe the point. Now consider Figure 10.4b, which shows two new axes, u_1 and u_2. Again, the ordered pair (u_1, u_2) is adequate to exactly describe the point, but u_2 is (compared to u_1) very nearly zero, most of the time. Thus, we would lose very little if we simply discarded u_2 and used the scalar, u_1, to describe each point. Our objective in this



(a) A region that is approximately an ellipse.



(b) A new set of coordinates, derived by a rotation of the original coordinates, in which one coordinate well represents the data.

Figure 10.4 Two ways of representing the same shape.

section will be to learn how to determine u_1 and u_2 in an optimal manner for an arbitrary distribution of points.

10.3.1 Derivation of the K-L Expansion

Let $\mathbf{x}$ be a d-dimensional random vector. We will describe $\mathbf{x}$ in terms of a set of basis vectors. That is, represent $\mathbf{x}$ by

$$\mathbf{x} = \sum_{i=1}^{d} y_i \mathbf{b}_i. \tag{10.9}$$

Here, the vectors $\mathbf{b}_i$ are deterministic (and may in general, be specified in advance). If any random vector $\mathbf{x}$ may be expressed in terms of the same d vectors, $\mathbf{b}_i (i = 1, \ldots, d)$, we say the vectors $\mathbf{b}_i$ *span the space* containing $\mathbf{x}$, and refer to them as a *basis set* for the set of all $\mathbf{x}'s$. To make further use of the basis set, we require[2]:

1. The $\mathbf{b}_i$ vectors are linearly independent.
2. The $\mathbf{b}_i$ vectors are orthonormal, i.e.,

$$\mathbf{b}_i^{\mathrm{T}} \mathbf{b}_j = \begin{cases} 1 & (i = j) \\ 0 & (i \neq j) \end{cases}. \tag{10.10}$$

Under these conditions, the y_i of Eq. 10.9, can be determined by

$$y_i = \mathbf{b}_i^{\mathrm{T}} \mathbf{x} \quad (i = 1, \ldots, d). \tag{10.11}$$

Here, we say the scalar y_i results from projecting $\mathbf{x}$ onto the basis vector $\mathbf{b}_i$ and we define

$$\mathbf{y} = [y_1, \ldots, y_d]^{\mathrm{T}}. \tag{10.12}$$

Suppose we wish to ignore all but m ($m < d$) components of $\mathbf{y}$, which are denoted the *principal components*, and yet we wish to still represent $\mathbf{x}$, although with some error. We

[2] To be a basis, the vectors do not have to be orthonormal, just not parallel; however, in this derivation, we will require orthonormality.

will thus calculate (by projection onto the basis vectors) the first m elements of $\mathbf{y}$, and replace the others with constants, forming the estimate.

$$\hat{\mathbf{x}} = \sum_{i=1}^{m} y_i \mathbf{b}_i + \sum_{i=m+1}^{d} \alpha_i \mathbf{b}_i. \tag{10.13}$$

The error that has been introduced by using some arbitrary constants, the α's of Eq. 10.13, rather than the elements of $\mathbf{y}$, is given by

$$\begin{aligned} \Delta\mathbf{x} &= \mathbf{x} - \hat{\mathbf{x}} \\ &= \mathbf{x} - \sum_{i=1}^{m} y_i \mathbf{b}_i - \sum_{i=m+1}^{d} \alpha_i \mathbf{b}_i, \end{aligned} \tag{10.14}$$

but since

$$\mathbf{x} = \sum_{i=1}^{m} y_i \mathbf{b_i} + \sum_{m+1}^{d} y_i \mathbf{b_i} \tag{10.15}$$

we arrive at

$$\Delta\mathbf{x} = \sum_{i=m+1}^{d} (y_i - \alpha_i)\mathbf{b}_i. \tag{10.16}$$

If we think of $\mathbf{x}$ and therefore $\Delta\mathbf{x}$ as random vectors, we can use the expected magnitude of $\Delta\mathbf{x}$ to quantify how well our representation works.

$$\epsilon(m) = E\left\{ \sum_{i=m+1}^{d} \sum_{j=m+1}^{d} (y_i - \alpha_i)\mathbf{b}_i^{\mathrm{T}} (y_j - \alpha_j)\mathbf{b}_j \right\} \tag{10.17}$$

$$= E\left\{ \sum_{i=m+1}^{d} \sum_{j=m+1}^{d} (y_i - \alpha_i)(y_j - \alpha_j)\mathbf{b}_i^{\mathrm{T}} \mathbf{b}_j \right\} \tag{10.18}$$

noting that y_i is a scalar, and recalling Eq. 10.10, this becomes

$$\epsilon(m) = \sum_{m+1}^{d} E\{(y_i - \alpha_i)^2\}. \tag{10.19}$$

To find the optimal choice for α_i, minimize with respect to α_i

$$\frac{\partial \epsilon}{\partial \alpha_i} = \frac{\partial}{\partial \alpha_i} E\{(y_i - \alpha_i)^2\} = E\{-2(y_i - \alpha_i)\}, \tag{10.20}$$

which can be set to zero, resulting in

$$\alpha_i = E\{y_i\} = \mathbf{b}_i^{\mathrm{T}} E\{\mathbf{x}\}. \tag{10.21}$$

So, we should replace those elements of $\mathbf{y}$ that we do not measure by their expected values. Something both mathematically and intuitively appealing.

Substituting Eq. 10.21 into Eq. 10.19, we have

$$\epsilon(m) = \sum_{i=m+1}^{d} E\{(y_i - E\{y_i\})^2\}. \tag{10.22}$$

Substituting Eq. 10.11 into Eq. 10.22:

$$\epsilon(m) = \sum_{i=m+1}^{d} E\{(\mathbf{b}_i^{\mathrm{T}}\mathbf{x} - E\{\mathbf{b}_i^{\mathrm{T}}\mathbf{x}\})^2\} \tag{10.23}$$

$$= \sum_{i=m+1}^{d} E\{(\mathbf{b}_i^{\mathrm{T}}\mathbf{x} - E\{\mathbf{b}_i^{\mathrm{T}}\mathbf{x}\})(\mathbf{b}_i^{\mathrm{T}}\mathbf{x} - E\{\mathbf{b}_i^{\mathrm{T}}\mathbf{x}\})\} \tag{10.24}$$

$$= \sum_{i} E\{\mathbf{b}_i^{\mathrm{T}}(\mathbf{x} - E\{\mathbf{x}\})(\mathbf{x}^{\mathrm{T}} - E\{\mathbf{x}^{\mathrm{T}}\})\mathbf{b}_i\} \tag{10.25}$$

$$= \sum_{i} \mathbf{b}_i^{\mathrm{T}}(E\{(\mathbf{x} - E\{\mathbf{x}\})(\mathbf{x} - E\{\mathbf{x}\})^{\mathrm{T}}\})\mathbf{b}_i \tag{10.26}$$

and we now recognize the term between the $\mathbf{b}'s$ in Eq. 10.26 as the covariance of $\mathbf{x}$:

$$\epsilon(m) = \sum_{i=m+1}^{d} \mathbf{b}_i^{\mathrm{T}} K_{\mathbf{x}} \mathbf{b}_i. \tag{10.27}$$

We could differentiate Eq. 10.27, set the result to zero, and attempt to find the vectors $\mathbf{b}_i$, but it wouldn't work because the $\mathbf{b}_i$'s that minimize the function are the zero vectors. We must make use of the fact that they are unit vectors. That is for every i, $\mathbf{b}_i^{\mathrm{T}}\mathbf{b}_i = 1$. We make that a constraint by using Lagrange multipliers.

Recall that an objective function can be minimized subject to a constraint by adding a term that is equal to zero, multiplied by a scalar *Lagrange multiplier*. For our problem, the objective function becomes

$$\sum_{i} \left[\mathbf{b}_i^{\mathrm{T}} K_{\mathbf{x}} \mathbf{b}_i - \lambda\left(\mathbf{b}_i^{\mathrm{T}}\mathbf{b}_i - 1\right)\right]. \tag{10.28}$$

To avoid confusion with the index of the summation, take the derivative with respect to some arbitrary $\mathbf{b}$, say $\mathbf{b}_k$. Doing that, one realizes that the only terms in the summation over i that are nonzero are the terms involving k, so the summation goes away, and we have

$$\nabla_{\mathbf{b}_i}\epsilon = 2K_{\mathbf{x}}\mathbf{b}_k - 2\lambda_k I\mathbf{b}_k. \tag{10.29}$$

Rearranging, this becomes

$$K_{\mathbf{x}}\mathbf{b}_k = \lambda_k \mathbf{b}_k. \tag{10.30}$$

That is, the optimal basis vectors are the eigenvectors of $K_{\mathbf{x}}$.

The covariance of $\mathbf{y}$ can be easily related to $K_{\mathbf{x}}$ by:

$$K_{\mathbf{y}} = E\{(\mathbf{y} - E\{\mathbf{y}\})(\mathbf{y} - E\{\mathbf{y}\})^{\mathrm{T}}\} = B^{\mathrm{T}} K_{\mathbf{x}} B \tag{10.31}$$

where the matrix B has columns made from the basis vectors, $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_d\}$.

Furthermore, in the case that the columns of B are the eigenvectors of $K_{\mathbf{x}}$, then B will be the transformation that diagonalizes K_x, resulting in

$$K_y = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \lambda_d \end{bmatrix}. \tag{10.32}$$

Substituting Eq. 10.32 into Eq. 10.27, we find

$$\epsilon(m) = \sum_{i=m+1}^{d} \mathbf{b}_i^{\mathrm{T}} \lambda_i \mathbf{b}_i. \tag{10.33}$$

Since λ_i is scalar,

$$\epsilon(m) = \sum_{i=m+1}^{d} \lambda_i \mathbf{b}_i^{\mathrm{T}} \mathbf{b}_i. \tag{10.34}$$

and, again remembering the orthonormal conditions on $\mathbf{b}_i$,

$$\epsilon(m) = \sum_{i=m+1}^{d} \lambda_i. \tag{10.35}$$

Thus, we represent a d-dimensional measurement, $\mathbf{x}$, by an m-dimensional vector, $\mathbf{y}$, where $m < d$,

$$y_i = \mathbf{b}_i^{\mathrm{T}} \mathbf{x}, \tag{10.36}$$

and the $\mathbf{b}_i$ are the eigenvectors of the covariance of $\mathbf{x}$. This expansion of a random vector in terms of the eigenvectors of the covariance matrix is referred to as the Karhunen-Loève expansion, or the "K-L expansion."

10.3.2 Properties of the K-L Expansion

Without loss of generality, the eigenvectors $\mathbf{b}_i$ can be sorted in terms of their corresponding eigenvalues. That is, assign subscripts to the eigenvalues such that

$$\lambda_1 > \lambda_2 > \dots > \lambda_d. \tag{10.37}$$

Then, we refer to $\mathbf{b}_1$, corresponding to λ_1, as the "major eigenvector."

Consider a Gaussian in two dimensions:

$$G(x, y) = \frac{1}{2\pi |K|} \exp\left(-\frac{[x\, y] K^{-1} \begin{bmatrix} x \\ y \end{bmatrix}}{2} \right), \tag{10.38}$$

and consider the C level set of this Gaussian, $G(x, y) = C$.

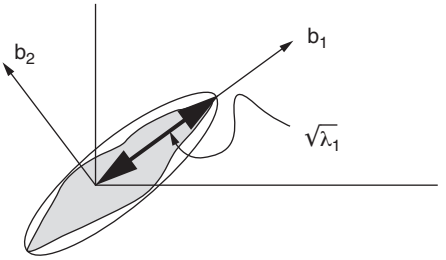


Figure 10.5 A covariance matrix may be thought of as representing a hyperellipsoid, oriented in the directions of the eigenvectors, and with extent in those directions equal to the square root of the corresponding eigenvalues.

Now, we take the log of both sides and we get

$$\ln C = \ln\left(\frac{1}{2\pi|K|}\right) - \frac{[x\,y]K^{-1}\begin{bmatrix}x\\y\end{bmatrix}}{2}. \tag{10.39}$$

Since $|K|$ is a scalar constant, we lump all the constants together into a new constant, $C' = \ln C - \ln(\frac{1}{2\pi|K|})$, and produce

$$C' = [x\,y]K^{-1}\begin{bmatrix}x\\y\end{bmatrix}. \tag{10.40}$$

Now, expand the inverse covariance matrix, using $K^{-1} = \begin{bmatrix}k_{11} & k_{12}\\ k_{21} & k_{22}\end{bmatrix}$, and we obtain

$$C' = k_{11}x^2 + 2k_{12}xy + k_{22}y^2. \tag{10.41}$$

To show that Eq. 10.41 represents an ellipse and not some other second order function, we must examine the covariance matrix more closely. We do an eigenvalue decomposition of $K^{-1} = E\Lambda E^{\mathrm{T}}$. In this decomposition, Λ is diagonal with the eigenvalues on the diagonal. If the eigenvalues are both positive, this will represent an ellipse. Since a covariance matrix will always have positive eigenvalues, then Eq. 10.41 represents an ellipse.

In higher dimensions, the same arguments lead to the conclusion that a level set of a Gaussian will be an ellipsoid or hyperellipsoid.

If we think of the distribution of points, $\mathbf{x}$, as represented by a hyperellipsoid, the major axis of that ellipsoid will pass through the center of gravity of the data, and will be in the direction of the eigenvector corresponding to the largest eigenvalue of $K_{\mathbf{x}}$. This is illustrated in Figure 10.5. Thus, the K-L transform fits an ellipse to two-dimensional data, an ellipsoid to three-dimensional data, and a hyperellipsoid to higher dimensional data.

Use in Straight Line Fitting

Consider a set of instances of the random vector $\mathbf{x}$, $\{\mathbf{x}_i\}$ $i = 1, \ldots, m$. We wish to find the straight line that best fits this set of data. Move the origin to the center of gravity of the set. Then, characterize the (currently unknown) best fitting line by its unit normal vector, $\mathbf{n}$. Then, for each point $\mathbf{x}_i$, the perpendicular distance from $\mathbf{x}_i$ to the best fitting line will be equal to the projection of that point onto $\mathbf{n}$, as illustrated in Figure 10.6. Denote this

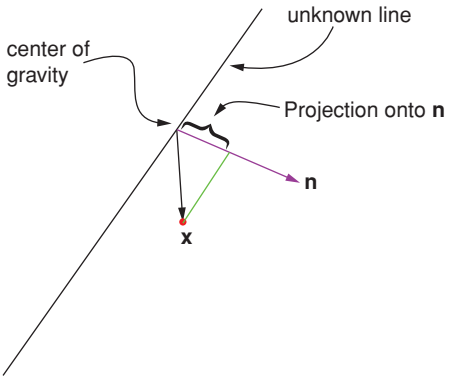


Figure 10.6 Minimize the sum of projections onto the normal.

distance by $d_i(\mathbf{n})$:

$$d_i(\mathbf{n}) = \mathbf{n}^{\mathrm{T}}\mathbf{x}_i. \tag{10.42}$$

To find the best fitting straight line, minimize the sum of the squared perpendicular distances,

$$\epsilon = \sum_{i=1}^{m} d_i^2(\mathbf{n}) = \sum_{i=1}^{m} (\mathbf{n}^{\mathrm{T}}\mathbf{x}_i)^2 = \sum_{i=1}^{m} (\mathbf{n}^{\mathrm{T}}\mathbf{x}_i)(\mathbf{x}_i^{\mathrm{T}}\mathbf{n}) = \mathbf{n}^{\mathrm{T}} \left(\sum_{i=1}^{m} \mathbf{x}_i\mathbf{x}_i^{\mathrm{T}} \right) \mathbf{n} \tag{10.43}$$

subject to the constraint that $\mathbf{n}$ is a unit vector,

$$\mathbf{n}^{\mathrm{T}}\mathbf{n} = 1. \tag{10.44}$$

Performing the constrained minimization using a Lagrange multiplier requires minimizing

$$\mathbf{n}^{\mathrm{T}} \left(\sum_{i=1}^{m} \mathbf{x}_i\mathbf{x}_i^{\mathrm{T}} \right) \mathbf{n} - \lambda(\mathbf{n}^{\mathrm{T}}\mathbf{n} - 1). \tag{10.45}$$

Defining $S = \sum_{i=1}^{m} \mathbf{x}_i\mathbf{x}_i^{\mathrm{T}}$, differentiate as

$$\frac{\partial \epsilon}{\partial \mathbf{n}} \left(\mathbf{n}^{\mathrm{T}}S\mathbf{n} - \lambda(\mathbf{n}^{\mathrm{T}}\mathbf{n} - 1) \right). \tag{10.46}$$

Differentiating the quadratic form $\mathbf{n}^{\mathrm{T}}S\mathbf{n}$, we get $2S\mathbf{n}$, and setting the derivative to zero results in

$$2S\mathbf{n} - 2\lambda\mathbf{n} = 0 \tag{10.47}$$

which is the same eigenvalue problem mentioned earlier. Thus we may state: The best fitting straight line passes through the mean of the set of data points, and will lie in the direction corresponding to the major eigenvector of the covariance of the set.

We have now seen two different ways to find the straight line that best fits the data: The method of least squares described in section 5.4.2, if applied to fitting a line rather than a plane, minimizes the vertical distance from the data points to the line. The method described in this section minimizes the perpendicular distance described by Eq. 10.43. The

method presented here is applicable to a line in any number of dimensions, and does not fail when the best fitting line happens to be vertical.

Other methods for fitting straight lines also exist. For example, [10.26] describes piecewise representations that preserve moments up to an arbitrarily specified order. Fitting functions to data occurs in many contexts. For example, Gorman [10.28] has looked at fitting not only straight edges, but points, straight lines, and regions with straight edges. By so doing, sub-pixel precision can be obtained.

10.3.3 Groups

This section covers a bit more terminology of transformations, moving from linear transformations to more general types.

As we discuss a bit more about the theory of transformations, one particular transformation, rotation in the plane, will be used as an example, as its properties are easy to visualize. Recall from section 10.2 that a rotation of a point in 2-space may be easily computed by a 2×2 matrix. This collection of such 2×2 matrices that do rotations is named $SO(2)$, the *special orthogonal group* in two dimensions. The columns of these matrices, when considered as vectors, are orthogonal, and the determinant is 1.0.

A *group*, in mathematics, is a set of transformations that satisfy four properties:

Identity The transformation that takes any member of the set to itself must itself be in the set. In $SO(2)$, the transformation $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ does this.

Inverse If a transform is in the set, its inverse must also be in the set. For example, if $\begin{bmatrix} 0.866 & .5 \\ -.5 & 0.866 \end{bmatrix}$ is in the set of transformations, then $\begin{bmatrix} 0.866 & -.5 \\ .5 & 0.866 \end{bmatrix}$ must also be in the set.

Associative Let A, B, and C be transformations in the set, and let D be the composition $D = B \circ C$, and $E = A \circ B$, where $\circ$ here denotes matrix multiplication. Then the following relation must hold:

$$A \circ D = E \circ C.$$

Said another way, $A \circ (B \circ C) = (A \circ B) \circ C$, that has the form of the familiar associative property. Here, since SO(2) is a linear group, the composition of transforms, $\circ$, may be computed by simple matrix multiplication.

Closure If A and B are in the set, then $A \circ B$ is in the group. For example if $A = \begin{bmatrix} 0.866 & 0.5 \\ -0.5 & 0.866 \end{bmatrix}$ and $B = \begin{bmatrix} 0.9397 & -0.3420 \\ 0.3420 & 0.9397 \end{bmatrix}$, then $A \circ B = \begin{bmatrix} 0.9848 & 0.1737 \\ -0.1737 & 0.9848 \end{bmatrix}$, and $A \circ B$ is also an element of the group.

10.4 Features of Regions

In this section, we introduce some important features used to describe a region. We start from some simple properties of a region like perimeter, diameter, and thinness. We

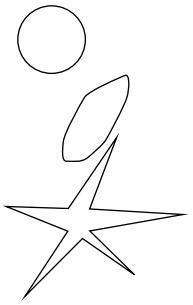


Figure 10.7 Results of applying two different compactness measures to various regions. Since a circle has the minimum perimeter for a given area, it minimizes T_a. A starfish on the other hand, has a large perimeter for the same area.

then extend the discussion to some invariant features like moments, chain codes, Fourier descriptors, and the medial axis.

10.4.1 Simple Features

In this section, we describe several simple features that can be used to describe the shape of a patch – the output of the segmentation process. Many of these features can be computed as part of the segmentation process itself. For example, since the connected component labeling program must touch every pixel in the region, it can easily keep track of the area. Following is a list of simple features that are likewise simple to calculate.

Average gray value In the case of black and white "silhouette" pictures, this is simple to compute.

Maximum gray value This is straightforward to compute.

Minimum gray value This is straightforward to compute.

Area (A) A count of all pixels in the region.

Perimeter (P) Several different definitions exist. Probably the simplest is a count of all pixels in the region that are adjacent to a pixel not in the region.

Diameter (D) The diameter is the length of the maximum chord; the distance between those two points in the region whose mutual distance is maximum [10.33, 10.35].

Thinness (also called Compactness[3]) (**T**) Two definitions for compactness exist: $T_a = \frac{P^2}{A}$ measures the ratio of the squared perimeter to the area; $T_b = \frac{D^2}{A}$ measures the ratio of the diameter to the area. Figure 10.7 compares these two measurements on example regions.

Center of gravity (CG) The center of gravity may be determined by

$$\hat{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i \tag{10.48}$$

for all N points in a region.

[3] Some authors [10.34] prefer not to confuse the mathematical definition of compactness with this definition, and thus refer to this measure as the isoparametric measure.

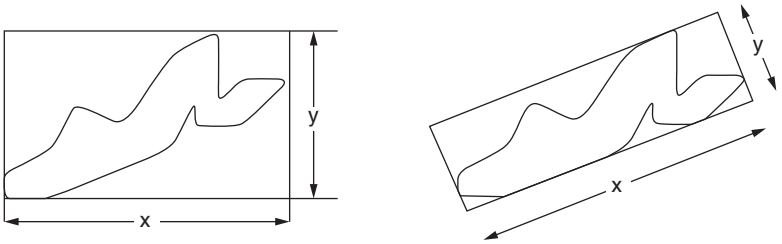


Figure 10.8 LEFT: y/x is the aspect ratio using one definition, with horizontal and vertical sides to the bounding rectangle. RIGHT: y/x is the minimum aspect ratio.

X-Y aspect ratio See Figure 10.8. The aspect ratio is the length/width ratio of the bounding rectangle of the region. This is simple to compute.

Minimum aspect ratio See Figure 10.8. This is again, a length/width ratio, but much more computation is required to find the minimum such rectangle, also called a *bounding box*[4]. The minimum aspect ratio can be a difficult calculation, since it requires a search for extremal points.

A very good approximation to the minimum aspect ratio can usually be obtained if we think of a region as represented by an ellipse-shaped distribution of points. In this case, as we showed in Figure 10.5, the eigenvalues of the covariance of points are measures of the distribution of points along the major and minor orthogonal axes. The ratio of those eigenvalues is quite a good approximation of the minimum aspect ratio. Furthermore the eigenvectors of the covariance are invariant to rotation and translation, and robust against noise.

Number of holes One feature that is very descriptive and reasonably easy to compute is the number of holes in a region. Connected components will yield an answer with minimal effort.

Triangle similarity Consider three points on the boundary of a region, P_1, P_2, and P_3, let $d(P_i, P_j)$ denote the Euclidean distance between two of those points, and let $S = d(P_1, P_2) + d(P_2, P_3) + d(P_3, P_1)$ be the perimeter of that triangle. The 2-vector

$$\left[\frac{d(P_1, P_2)}{S} \quad \frac{d(P_2, P_3)}{S}\right] \tag{10.49}$$

is simply the ratio of side lengths to perimeter. It is invariant to rotation, translation, and scale change [10.7].

Symmetry In two dimensions, a region is said to be "mirror-symmetric" if it is invariant under reflection about some line. That line is referred to as the axis of symmetry. A region is said to have rotational-symmetry of order n if it is invariant under a rotation of about some $\frac{2\pi}{n}$ point, usually the center of gravity of the region. There are two challenges in determining the symmetry of regions. The first is simply determining the axis. The other is to answer the question "how symmetric is it?" Most papers prior to 1995 that analyzed symmetry of regions in Computer Vision applications treated symmetry as a predicate: either the region is symmetric or it is not. Zabrodsky et al.

[4] The methods of section 10.3.2 give a simple way to estimate the aspect ratio.

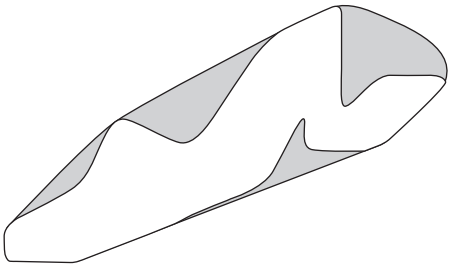


Figure 10.9 Convex hull of a region. The shaded area is the convex discrepancy.

[10.19, 10.43] present a measure called the symmetry distance that quantifies *how* symmetric a region is.

The Convex Discrepancy If one were to stretch a rubber band around a given region, the region that would result would be the convex hull (Figure 10.9). The difference in area between a region and its convex hull is the convex discrepancy. See Shamos [10.33] for fast algorithms for computing the convex hull, and [10.14] for such algorithms for parallel machines.

If we are fortunate enough to have a convex region to analyze, we can find the diameter in $O(n \log n)$ time, by finding the convex hull first. Furthermore, finding the convex hull provides us with another simple feature, the convex discrepancy, as illustrated in Figure 10.9.

10.4.2 Moments

The moments of a shape are easily calculated, and as we shall see, can be made robust to similarity transforms. A moment of order $p + q$ may be defined on a region as

$$m_{pq} = \int x^p y^q f(x, y) dx dy \tag{10.50}$$

where the integral is over the entire region, not just the boundary. If the function f is 1 inside the region and 0 outside, the area of the region is then m_{00}, and we find that the center of gravity is

$$m_x = \frac{m_{10}}{m_{00}} \quad m_y = \frac{m_{01}}{m_{00}}. \tag{10.51}$$

Invariance to Translation

Although the results of Eqs. 10.50 and 10.51 are totally dependent on the coordinate system in which the measurements are made, it is possible to derive a set of moment-like measurements that are invariant to translation by moving the origin to the center of gravity.

$$\mu_{pq} = \int (x - m_x)^p (y - m_y)^q f(x, y) dx dy. \tag{10.52}$$

These are called the *central moments*.

Since the central moments are calculated relative to the center of the region, the central moments are obviously invariant to translation.

Table 10.1 The Hu invariant moments

$$\phi_1 = \eta_{20} + \eta_{02}$$
$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$
$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$
$$\phi_4 = (\eta_{30} - \eta_{12})^2 + (\eta_{21} - \eta_{03})^2$$
$$\phi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{03} + \eta_{21})^2] + (3\eta_{21} - \eta_{03})(\eta_{03} + \eta_{21})[3(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2]$$
$$\phi_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{03} + \eta_{21})$$
$$\phi_7 = (3\eta_{12} - \eta_{30})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2] + (3\eta_{12} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2]$$

Almost always, we are interested in region shape rather than brightness distribution, so we usually set $f = 1$.

Invariance to Scale Change

Suppose a region is zoomed by moving it closer to the camera. As discussed in section 5.4.6, this change is referred to as a *scale change*. Then the points after the change (assuming the scale change in the x and y directions are the same) are related to the points before the change by the relationship

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \tag{10.53}$$

Invariance to scale can be accomplished by normalization as follows: When an image is scaled by α, the central moment becomes

$$\eta_{pq} = \int (\alpha x)^p (\alpha y)^q d(\alpha x) d(\alpha y) = \alpha^{p+q+2} \mu_{pq}. \tag{10.54}$$

To find α, we require that the area of the scaled region, that is η_{00}, always be one. In that case, $\eta_{00} = \alpha^2 \mu_{00} = 1$, and therefore, $\alpha = \mu_{00}^{-1/2}$. Substituting that into Eq. 10.54,

$$\eta_{pq} = \alpha^{p+q+2} \mu_{pq} = \left(\mu_{00}^{-1/2}\right)^{p+q+2} \mu_{pq}. \tag{10.55}$$

Finally the normalized central moment can be written as

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \tag{10.56}$$

where $\gamma = \frac{p+q+2}{2}$ will lead to moments invariant to translation and scale changes.

The Hu moments [10.15] also incorporate rotation to find a set of moments that have no orders higher than 3 and are invariant to translation, rotation, and scale change, which means that we get the same moment, even though the image may be moved, rotated, or have its scale changed. They are listed in Table 10.1.

It is interesting to observe that although the theory of invariant moments was developed by Hu in 1962, they were made most popular by publication in the 1977 book by Gonzalez and Wintz [10.12].

Since their original development by Hu [10.15], the concept has been extended to moments that are invariant to affine transforms by Rothe et al. [10.30]. Despite their

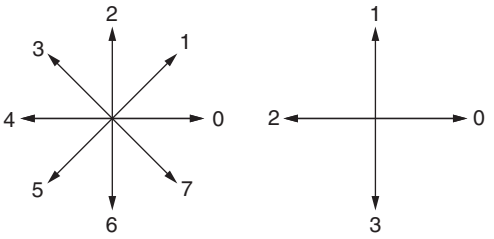


Figure 10.10 The eight directions (for 8-neighbor) and four directions (for 4-neighbor) in which one might step in going from one pixel to another around a boundary.

attractiveness, strategies based on moments do have problems, not the least of which is sensitivity to quantization and sampling [10.24]. (See Assignments for this chapter).

The use of moments is actually a special case of a much more general approach to image matching [10.30] referred to as the method of normalization. In this approach, we first transform the region into a canonical frame by performing a (typically linear) transform on all the points. The simplest such transformation is to subtract the coordinates of the center of gravity (CG) from all the pixels, thus moving the coordinate origin to the CG of the region. In the more general case, such a transformation might be a general affine transform, including translation, rotation, and shear. We then do matching in the transform domain, where all objects of the same class (e.g., triangles) look the same.

Some refinements are also required if moments are to be calculated with grayscale images, that is, when the f of Eq. 10.50 is not simply a one or a zero. All the theory of invariance still holds, but as Gruber and Hsu [10.13] point out, noise corrupts moment features in a data-dependent way.

Once a program has extracted a set of features, this set can be used either to match two observations or to match an observation to a model. The use of simple features in matching is described in section 10.5.1.

10.4.3 Chain Codes

A chain code is a feature describing the boundary of a region. In a chain code, a walk around a region is represented by a sequence of numbers, all between 0 and 7 (if using eight directions) or between 0 and 3 (if using four directions), designating the direction of each step. The eight and four cardinal directions are defined as illustrated in Figure 10.10. The boundary of a region may then be represented by a string of single digits. A more compact representation utilizes superscripts whenever a direction is repeated. For example, 0012112776660 could be written $0^2121^2276^2263^30$, and illustrates the boundary shown in Figure 10.11. The ability to describe boundaries with a sequence of symbols plays a significant role in the discipline known as *syntactic pattern recognition*, and appears in the Computer Vision literature. Such a representation allows recognition of shapes using principles from compiler theory.

10.4.4 Fourier Descriptors

In this section, we discuss a frequency-domain representation for shape. The title "Fourier Descriptor" is used for historical reasons. However, by the definition of this book,

Table 10.2 Equivalence between motions in the image and transform domains

In the image	In the transform
A change in size	multiplication by a constant
A rotation about the origin	phase shift (every term has the same amount added to its phase)
A translation	a change in the DC term

the algorithm is a shape representation method rather than a descriptor as defined in Chapter 11.

Given the boundary of a region, we imagine that the region is drawn on the complex plane, and the x coordinate of each point then represents a distance along the real axis. The y coordinate represents a distance along the imaginary axis. Each boundary point is therefore viewed as a single complex number. Traversing a closed boundary then results in a (cyclic) sequence of complex numbers.

Taking the Fourier transform of this sequence produces another sequence of complex numbers, one that can be shown to possess types of invariance as illustrated in Table 10.2. The following example illustrates these ideas in a somewhat oversimplified way: suppose we have the Fourier descriptors for two boundaries:

```
f1 = .7, .505, .304, .211, ...
f2 = .87, .505, .304, .211, ...
```

We see that these two sequences differ only in the first (DC) term. They therefore represent two encodings of the same boundary that differ only by a translation. This example is oversimplified, because in reality, the sequence will be a set of complex numbers, not real numbers as illustrated, but the concepts are identical.

Practical Considerations in Using Fourier Descriptors

How we represent the movement from one boundary point to another is critical. Simply using a 4-neighbor chain code produces poor results. Use of an eight-neighbor code reduces the error by 40 to 80%, but is still not as good as one would prefer. A superior method is to resample the boundary (see Sec. 10.6) so the pixels along the boundary are equally spaced.

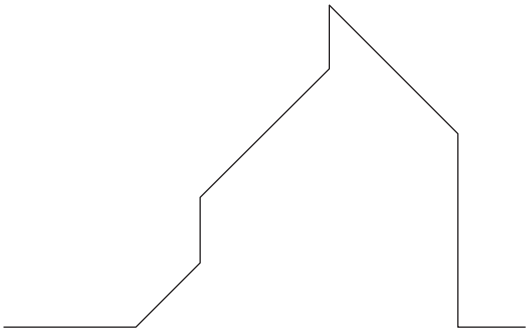


Figure 10.11 The boundary segment represented by the chain code 001211277660. (The validity of this code is a homework problem. It is either correct or almost correct).

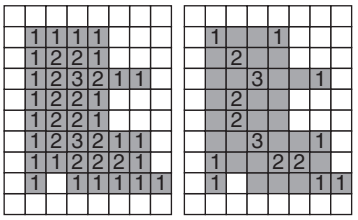


Figure 10.12 An example region whose DT, computed using 4-neighbors, is shown on the left. The morphological skeleton is illustrated on the right, and simply consists of the local maxima of the DT.

There are other complications as well, including the observation that the usual parameterization of the boundary (arc length) is not invariant to affine transformations [10.2, 10.42]. Experiments have [10.20] compared affine-invariant Fourier descriptors and autoregressive methods. For more detail on Fourier descriptors, see [10.3] and Assignment 10.5.

10.4.5 The Medial Axis

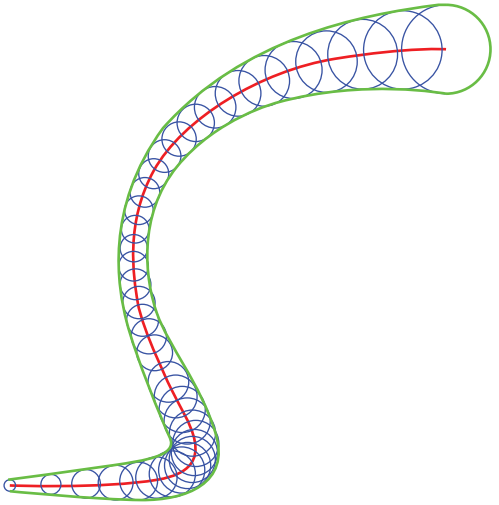
In two dimensions, the medial axis (also sometimes referred to as the *skeleton* of a region is defined as the locus of the centers of *maximal circles*. A maximal circle is the largest circle that can be located at a given point in the region. Let's say that a bit more carefully (we all need to learn how to use mathematics to make our wording precise): Maximal circles touch the region boundary at at least 2 points, such that no circle points are outside the region.

Any point on the medial axis can be shown to be a local maximum of a distance transform (DT). A point with DT values of k is a local maximum if none of its neighbors has a greater value. Figure 10.12 (left) repeats Figure 7.13. The set of local maxima of this distance transform is illustrated on the right.

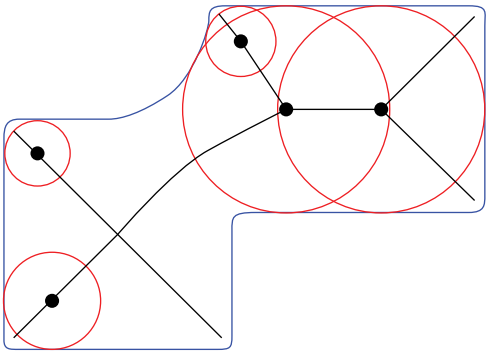
Examples of use of the medial axis are illustrated in Figure 10.13. Another way to think about the medial axis is as the minimum of an electrostatic potential field. This approach is relatively easy to develop if the boundary happens to be straight lines, or in three dimensions, planes [10.8]. See [10.11] for additional algorithms to efficiently compute the medial axis.

Relating the Medial Axis and Ridges

The medial axis is defined only for binary images, or at least only for images in which each region may be represented by its boundary. The ridge (recall section 4.3.3), however, is defined for grayscale images. We can relate these two characterizations using the following strategy: Use a scale-space representation of the image. That is, blur the binary image. This process turns the binary image into a gray-scale image. Now find the ridge. Start with the ridge at high scale (lots of blur). This is your initial estimate of the medial axis. Now, reduce the scale slightly and see what new ridges appear. Add them to your estimate. Continue this process over varying scales. See Pizer et al. [10.29] for more details on this philosophy, which is called the *Intensity Axis of Symmetry*.



(a) Medial axis (in red) of a curved planar tubular-like object. From F.E. Wolter: http://welfenlab.de/archive/mirror/brown00/figs.html.Used with permission.



(b) Medial axis used to generate ridge-lines for the rooves of buildings.From https://spacesymmetrystructure.wordpress.com/2009/10/05/medial-axes-voronoi-skeletons/.Used with permission.

Figure 10.13 Example shapes and their medial axes.

10.5 Matching Feature Vectors

Suppose a set of measurements was made about a region, e.g., area, perimeter, color, as we discussed in section 10.4.1. If these measurements were collected into a vector, that "feature vector" could be used to represent the region shape. Two regions could then be compared by computing some distance measure between their feature vectors. Thus, if g is the feature vector resulting from an image of a typical gasket, f the feature vector of a typical flange, and x the vector of an unknown, we might determine whether x is a gasket or a flange by determining if $||x - f|| < ||x - g||$.

10.5.1 Matching Simple Features

The most straightforward way to use the simple features that were described in section 10.4.1 is to use them in a pattern classifier. To do this, we will extract a statistical representation for the model and the object and match those representations. The strategy is as follows:

1. Decide which measurements you wish to use to describe the shape. For example, in a system that identifies an unknown shape as a flange or a gasket, one might build a system that measures seven invariant moments and the aspect ratio, for a total of eight "features." The best collection of features is application-dependent, and includes methods for optimally choosing feature sets that are beyond the scope of this book (see

[10.10, 10.36], which are just a few of many texts in statistical methods). Organize these eight features into a vector, $\mathbf{x} = [x_1, x_2, \ldots, x_8]^{\mathrm{T}}$.

2. Describe a "model" object using a collection of example images (called a "training set"), from which feature vectors have been extracted. Continuing the example with eight features, a set of n images of flanges would be collected and the feature vector of each flange measured. Then, the average feature vector for flanges would be computed.

   Gaskets would be similarly characterized by an average over a set of sample gaskets.
3. Now, given an unknown region, characterized by its feature vector $\mathbf{x}$, shape matching consists of finding the model that is "closest" in some sense to the observed region. We make the decision based on which of these distances is smaller.

In the next section, we discuss matching two vectors. In that section, an unknown vector of measurements is compared with the average of those measurements made on a training set.

10.5.2 Matching Vectors

Assuming vectors $\mathbf{a}$ and $\mathbf{b}$ are the same dimension, it makes sense to subtract them

$$\mathbf{d} = \mathbf{a} - \mathbf{b},$$

in the problem of finding matching descriptors in two images, (the correspondence problem). However, a more general problem is finding the class to which a vector belongs. In that case, we need to come up with a representation for the class, and that representation will be statistical.

10.5.3 Matching a Vector to a Class

If $\mathbf{d} = \mathbf{a} - \mathbf{b}$, then the Euclidean distance from $\mathbf{a}$ to $\mathbf{b}$ is $\sqrt{\mathbf{d}^{\mathrm{T}}\mathbf{d}}$. In the description below, we use the word *class* frequently. By that term, we mean that we are trying to decide whether an unknown feature vector is a member of one of several possible classes. For example, we might seek to determine if a shape in the image is the silhouette of a gasket or a flange.

If we have a large set of examples of flanges and a similarly large set of gaskets, and both classes can be represented by feature vectors, we can describe the classes by their statistics. For example, the *sample mean*

$$\mathbf{m}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} \mathbf{x}_{ij} \tag{10.57}$$

denotes the mean of class i, containing n_i example vectors, $\mathbf{x}_{ij}$. When a collection of points is drawn from the Gaussian distribution,

$$\frac{1}{(2\pi)^{d/2}|K|^{1/2}} \exp\left(-\frac{(\mathbf{x}-\mu)^{\mathrm{T}}K^{-1}(\mathbf{x}-\mu)}{2}\right), \tag{10.58}$$

it is not hard to show that the sample mean of those points is μ.

Given an unknown vector $\mathbf{x}$ and a collection of classes, each represented by its mean, $\{\mathbf{m}_1, \ldots, \mathbf{m}_i, \ldots, \mathbf{m}_c\}$ where c is the number of classes, we could calculate the Euclidean

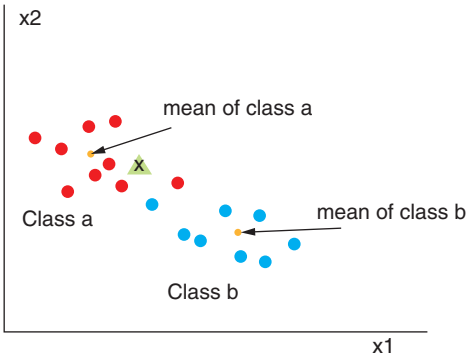


Figure 10.14 Two classes and their means are illustrated, one in red and one in blue. The unknown vector $\mathbf{x}$ is assigned to class a because it is closer to the mean of class a than the mean of class b.

distance from $\mathbf{x}$ to $\mathbf{m}_i$, and make the decision that $\mathbf{x}$ belongs to the closest class, that is, the class whose mean is closest to $\mathbf{x}$ as illustrated in Figure 10.14.

There isn't really any reason to calculate the square root, since the square is monotonic: $\mathbf{x}^{\mathrm{T}}\mathbf{x} > \mathbf{y}^{\mathrm{T}}\mathbf{y} \implies \sqrt{\mathbf{x}^{\mathrm{T}}\mathbf{x}} > \sqrt{\mathbf{y}^{\mathrm{T}}\mathbf{y}}$.

This technique is known as *Closest Mean Classification*, and is probably the simplest of the many statistical pattern recognition methods.

The idea of using the closest mean needs to be expanded, as illustrated in Figure 10.15, where two Gaussian probabilities are shown, illustrating the fact that the nearest mean algorithm will fail unless (co)variance is considered. Because of the need for considering the variance in our decision making, rather than the distance to the mean, $(\mathbf{x} - \mu_i)^{\mathrm{T}}(\mathbf{x} - \mu_i)$, we make use of the *Mahalanobis distance*, $(\mathbf{x} - \mu_i)^{\mathrm{T}}K_i^{-1}(\mathbf{x} - \mu_i)$.

Use of the Mahalanobis distance is just one of the many methods in the discipline of Statistical Pattern Recognition, a class every student reading this book should take.

10.6 Describing Shapes Using Boundaries

So far, we have seen many examples of shapes and techniques for representing them. You should have a pretty good grasp on what is meant by shape. Now, let's try to abstract our understanding and see what can be said about shapes in general.

First, we need to describe a concept that will occur many times in the discussion of boundaries, arc length. Imagine you are walking around a digital contour drawn in the x, y plane. As you step from pixel to pixel, sometimes you will travel a distance of 1.0, and sometimes a distance of 1.414. When thinking about a curve using continuous math, we think of a curve as a vector-valued function of a single parameter:

$$X(s) = \begin{bmatrix} x(s) \\ y(s) \end{bmatrix},$$

where s is the distance (the "arc length") along the curve from some defined starting point. We may need to perform integrals or other operations that assume the parameter, s is

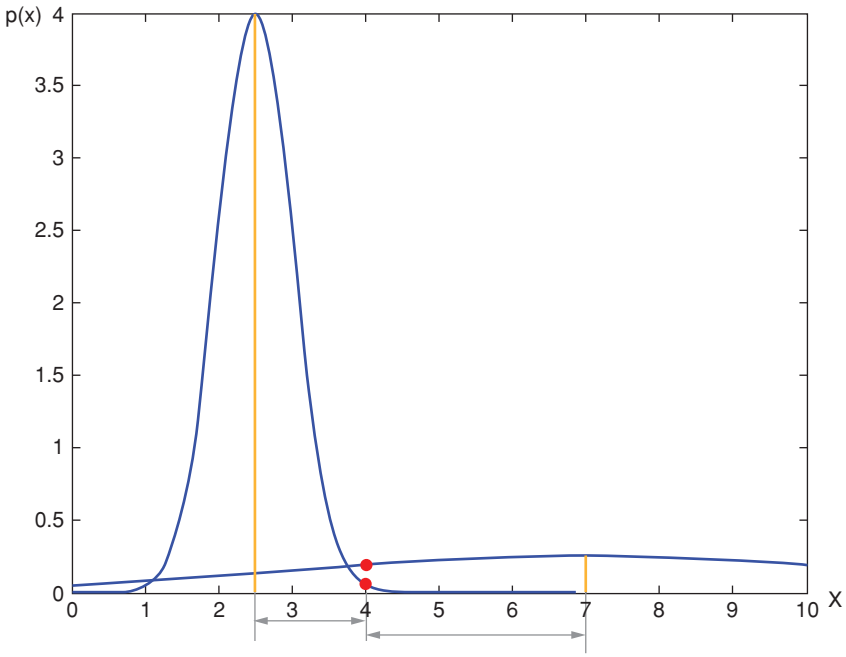


Figure 10.15 Two Gaussian distributions are illustrated, one with a mean of 2.5, and another with mean of 7. The two distributions have significantly different variances. The point $x = 4$ is closer to the mean of class 1 (with a distance of 1.5) than to the mean of class 2 (with a distance of 3), yet at x, the probability is higher that x is in class 2.

uniformly sampled. This requires resampling of the boundary, which in turn prohibits boundary points from falling exactly on pixel boundaries. Figure 10.16 illustrates a simple way to resample a boundary.

If the resampled interval is 1.0, then the number of new sample points will be equal to the perimeter of the region. In addition to making measures of perimeter consistent, arc length parameterization also provides a great deal of power to curve representations, which you will learn when you take a course in differential geometry (which is another of the courses advanced students in computer vision should definitely take).

10.6.1 The Shape Matrix

Consider the boundary of some region in an image. Write every point on that boundary as a 2-vector of the form $\begin{bmatrix} x \\ y \end{bmatrix}$. Assuming the boundary contains n points, the entire boundary can be written as a matrix of these vectors

$$X = \begin{bmatrix} x_1 & \dots & x_i & \dots & x_n \\ y_1 & \dots & y_i & \dots & y_n \end{bmatrix}. \tag{10.59}$$

For now, forget that this was the boundary of some region, and just think of it as a matrix representation for the shape X.

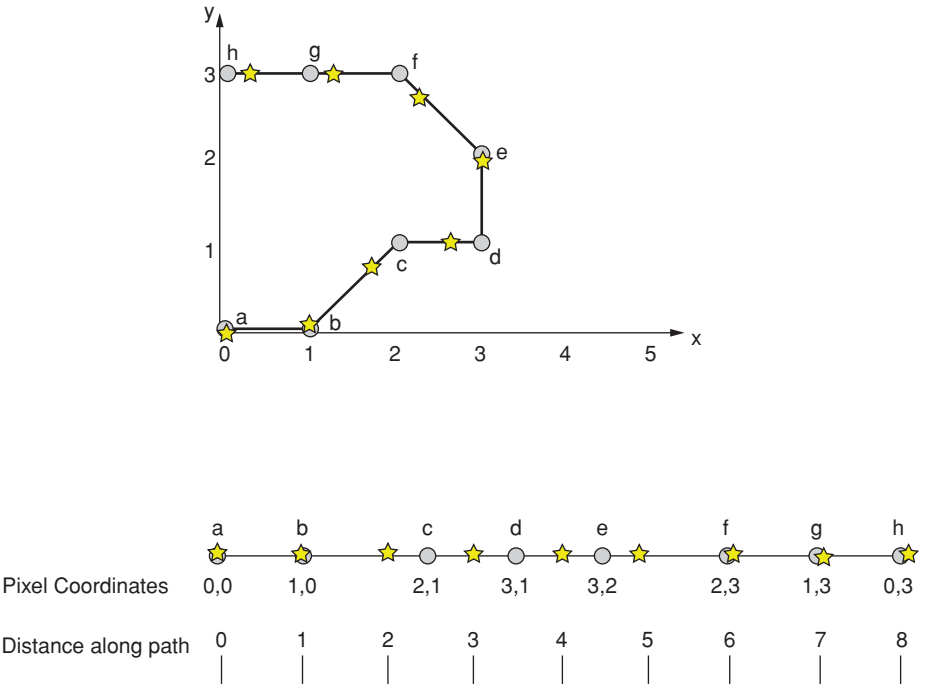


Figure 10.16 The process of resampling a boundary. Points located at integer points (pixels) on the boundary are marked with circles and labeled with letters. The distances between those points are found and laid end-to-end to form a horizontal line. That horizontal line is then resampled at equal intervals and the sample points marked with stars.

How many possible shapes are identical to X? It depends, of course, on what you mean by "identical." If two shapes are described as identical even if they are drawn at different points on the paper, then X has an infinite number of "twins" just due to translation. If the only difference is a rotation, or a scale change, another couple of infinities appear. This collection of infinite sets is called *shape space*.

Of course, in Computer Vision, our problem is the opposite. We are given some point in the shape space, and we seek the original shape. So we really just want to reduce these infinities to one. The components of the problem are broken down as follows:

Translation We compensate for translation as follows: find the center of gravity of X, $[\hat{x}, \hat{y}]^{\mathrm{T}}$, and subtract the vector $[\hat{x}, \hat{y}]^{\mathrm{T}}$ from each column of X. This produces a matrix X_t that is the same no matter where it is drawn.

Scale Change In the absence of occlusion, we compensate for scale change by calculating $\sqrt{Tr(XX^{\mathrm{T}})}$. We then divide each element of X_t by this scalar to produce X_{ts}, a matrix that is the same no matter where it is drawn and how much it is zoomed.

If the shape can be partially occluded, this simple normalization will not be sufficient. In section 10.6.4 we will discuss SKS, which can compensate for some partial occlusion.

Rotation Rotation becomes more challenging to deal with in a simple, yet very general way. See [10.21]. Rotation, however, can be dealt with easily if one simply uses features such as distances and curvatures that are already invariant to rotation.

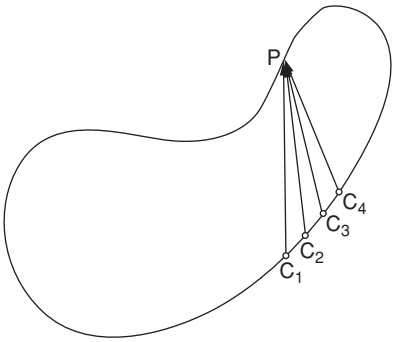


Figure 10.17 All the points on the boundary of a shape are denoted by $\{C_1, C_2, \ldots C_N\}$. The ordered set of vectors from all those points to a single one of the shape context points are calculated and used in determining the shape context.

10.6.2 The Shape Context

The Shape Context [10.5] [10.6] algorithm for matching 2D shapes was introduced by Belongie et al. in [10.4]. As illustrated in Figure 10.17, given a contour C, represented as an ordered set of points, $C = \{C_1, C_2, \ldots C_N\}$, let $P \in C$. For any element of C, we can compute the vector from that point to P, and thus can construct an ordered set of such vectors, $\Delta_P = \{P - C_1, P - C_2, \ldots, P - C_N\}$. The elements of Δ_P are converted to log-polar coordinates and coarsely[5] quantized, and then a two-dimensional histogram is constructed, defining the shape context of point P:

$$h(P, \theta, r) = \sum_j \delta(\theta - \theta_j, r - r_j) \qquad j = 1, \ldots, N, \tag{10.60}$$

where the δ denotes the Kronecker delta:

$$\delta(x, y) = \begin{cases} 1 & \text{if} \quad x = 0 \text{ and } y = 0 \\ 0 & \text{otherwise} \end{cases}.$$

The histogram is simply a count of how many times each particular (angle, log-distance) pair occurs on this contour. Since the histogram is of a fixed size, in this case, 60 bins, we may index it with a single index and denote it by $h(P, k)$. We refer to $h(P, k)$ as the *shape context* of point P in curve C.

Clearly, the choice of a reasonable resolution for the histogram depends on N.

The shape context for a particular contour is the collection of the shape contexts of all the points in that contour.

In the absence of occlusion, the shape context may be made invariant to similarity transforms [10.4]. Invariance to translation is automatic, since the Shape Context only uses relative distances and angles. Invariance to scale is accomplished by dividing all distances Δ_P by the median of all such distances in the shape. Since a log-polar system is used, the calculation used is actually a subtraction of logs.

[5] In [10.6] the histogram has 12 equally spaced angle bins and 5 equally spaced log-radius bins.

Matching Contours using Shape Context

To match region shapes, with the *shape context* we will use it to find a measure that will characterize how well curve C matches some other curve, say ${}^{j}C$. Let P denote a point on curve C and let $Q \in {}^{j}C$. Denote the corresponding histograms as $h(P, k)$ and $h(Q, k)$. For any particular P and Q, we may match their shape contexts by matching individual points along the two curves. Since P and Q denote points along specific curves, they may be thought of as indices. Since P indexes curve C, and Q indexes ${}^{j}C$, we can construct a matrix of matching costs. Specifically, if we believe point P in curve C is the same as point Q in curve ${}^{j}C$, we define a cost of assigning P to correspond to Q. This "assignment cost" is

$$\gamma_{PQ} = \frac{1}{2}\sum_{k=i}^{K}\frac{(h(P,k) - h(Q,k))^2}{h(P,k) + h(Q,k)}. \tag{10.61}$$

All these assignment costs may be calculated and entered into a matrix.

$$\Gamma_{ij} = \left[\gamma_{PQ}\right] P \in C, Q \in {}^{j}C. \tag{10.62}$$

The matrix Γ is a representation of the cost of matching shape i to shape j. if $N_i = N_j$, Γ_{ij} is square, if not, Γ_{ij} may be made square by adding "dummy" nodes with constant match cost. To extract a scalar measure of this cost, we must find the mapping $f : C \rightarrow {}^{j}C$ that is a bijection (1:1 and onto), and that minimizes the total cost. That is, to the P^{th} element of C we must find exactly one element of ${}^{j}C$ to which we assign it. This is referred to as the *linear assignment problem* (LAP). This problem may be stated in terms of the matrix Γ_{ij} by observing that, for each row, we must choose one and only one column for the match, and that column must, in turn match no other rows.

The linear assignment problem is one of a large set of linear programming problems that may be solved using the simplex algorithm (see [10.9]). The special case of the linear assignment problem may be solved in $O(n^3)$, using several strategies, the most well-known of which is the "Hungarian algorithm" [10.23]. We used the implementation by Jonker and Volgenant [10.17].

Assuming that unknown curve ${}^{i}C$ is being matched to a data base of models, B. The solution to the LAP produces a number, $\hat{\Gamma}_{ij} = \min_j(\Gamma_{ij})$, ${}^{j}C \in B$, which is the best assignment possible for these two curves. Thus the index of the best match $m = \text{argmin}_j(\hat{\Gamma}_{ij})$.

10.6.3 Curvature Scale Space

Curvature Scale Space (CSS) is a shape representation method introduced by Mokhtarian and Macworth [10.25]. CSS has also been adopted in the MPEG-7 standard as a contour shape descriptor [10.44]. The CSS representation is a multi-scale organization of curvature zero crossing points of a planar curve. The representation is defined in the continuous domain and sampled later. Consider a curve parametrized by arc length s, and s is set to lie in the range $0 \leq s \leq 1$, to provide scale invariance, then

$$C(s) = [x(s), y(s)]^{\mathrm{T}}. \tag{10.63}$$

The curvature of such a curve is given by:

$$\kappa(s) = \frac{(\dot{x}(s)\ddot{y}(s) - \ddot{x}(s)\dot{y}(s))}{(\dot{x}(s)^2 + \dot{y}(s)^2)^{3/2}}, \tag{10.64}$$

where $\dot{x}$ denotes $\frac{\partial x}{\partial s}$.

The above contour can be successively blurred by convolving it with a 1D Gaussian kernel of width σ, where the scale (σ) is increased at each level of blurring. Let this blurred version of the contour be given by $C : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R} \times \mathbb{R}$

$$C(s, \sigma) = (x(s) * G(s, \sigma), y(s) * G(s, \sigma)). \tag{10.65}$$

The first and second derivatives of the curve at each scale σ can be estimated as follows:

$$\begin{aligned} x_s(s, \sigma) &= x(s) * G_s(s, \sigma) \\ y_s(s, \sigma) &= y(s) * G_s(s, \sigma) \\ x_{ss}(s, \sigma) &= x(s) * G_{ss}(s, \sigma) \\ y_{ss}(s, \sigma) &= y(s) * G_{ss}(s, \sigma) \end{aligned} \tag{10.66}$$

where the subscript denotes partial differentiation.

The curvature at each scale σ can be found by making the dependence on scale explicit as:

$$\kappa(s, \sigma) = \frac{(x_s(s, \sigma)y_{ss}(s, \sigma) - x_{ss}(s, \sigma)y_s(s, \sigma)}{(x_s(s, \sigma)^2 + y_s(s, \sigma)^2)^{3/2}}. \tag{10.67}$$

The curvature zero-crossings are the points on the contour where the curvature changes sign. The CSS representation computes the points of curvature zero-crossings at each scale and represents them in the (s, σ) plane. This is shown in Figure 10.18 with arc length(s) along the horizontal axis and the scale (σ) along the vertical axis. This is called the CSS image. The contour of the fish was obtained from the SQUID data base.[6] We implemented a version of CSS and applied it to obtain the right-hand figure.

The set of maxima of the CSS image is used as the shape signature for the given contour. Maxima with low scale values in the CSS image are related to the noise in the curve and are ignored. Following Mokhtarian [10.1], we assume anything below $\frac{\sigma_{max}}{6}$ is considered to be noise and is not used in the matching process, where σ_{max} is the highest maximum in the CSS image.

CSS descriptors are translation-invariant because of the use of curvature. Scale invariance is obtained by resampling the curve to a fixed number[7] of boundary points. Rotation and starting point changes cause circular shifts in the CSS image and are compensated for during the matching process, described later in this section.

Matching Contours with Curvature Scale Space Matching

Earlier in this section the curvature scale space was defined. Here, we discuss the use of this representation in matching plane curves.

[6] `http://www.ee.surrey.ac.uk/CVSSP/demos/css/demo.html`.

[7] We recommend resampling the contour to 200 points.

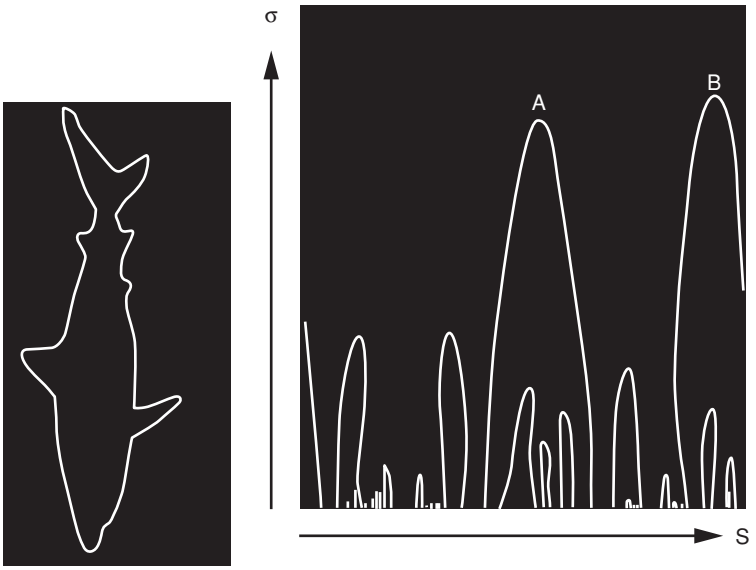


Figure 10.18 A fish contour from the SQUID data base [10.1, 10.25] and the corresponding CSS image. The vertical axis in the right-hand image is scale.

The figure on the right of Figure 10.18 illustrates a curvature scale space with two high peaks, marked by letters. The horizontal axis, s, represents arc length between zero and one as the curve is traversed. The vertical axis is scale. The peak marked with B indicates that somewhere near the end of the contour the curvature changes sign with a high scale.

The curvature scale space matching algorithm is described in detail in [10.25] [10.1]. Every curve in their database is represented by a set $\{(s, \sigma)\}$ of pairs, each denoting the arc length and scale of a maximum[8] in the CSS image. The matching process involves finding correspondences between the maxima of the two contours. We will refer to the two images as the target and the model.

Let $M_{target} = \{(t_1, \gamma_1), (t_2, \gamma_2), \ldots, (t_L, \gamma_L)\}$ be the maxima of the target, parametrized by arc length t and arranged in descending order of scale γ. Similarly, let $M_{model} = \{(s_1, \sigma_1), (s_2, \sigma_2), \ldots, (s_N, \sigma_N)\}$ be the maxima of the model, parametrized by arc length s and arranged in descending order of scale σ. The matching algorithm is as follows:

1. The highest maximum in the target and the model CSS image are used to find the CSS shift parameter.

$$\alpha = s_1 - t_1 \tag{10.68}$$

   This will compensate for any changes in the starting point and orientation of the contour. Two lists are initialized: one with the maximum pair (t_1, γ_1) from the target and the other with the maximum pair (s_1, σ_1) from the model. The cost for the match is initialized as:

$$MC = |\sigma_1 - \gamma_1| \tag{10.69}$$

[8] Here, *maxima* means the point of highest scale.

2. Now, for the next highest maximum (t_2, γ_2) in the target, we apply the CSS shift parameter (α) calculated earlier. We find the closest maximum in the model to this shifted target maximum that is not present in the model list.

$$(s_m, \sigma_m) = \arg \min_{i=1,\ldots,N} ||(s_i, \sigma_i) - (t_2 + \alpha, \gamma_2)||, (s_i, \sigma_i) \notin M_{model} \tag{10.70}$$

The two maxima are added to their respective lists. The cost of the match is updated as follows:

$$MC = \begin{cases} MC + ||(s_m, \sigma_m) - (t_2 + \alpha, \gamma_2)|| & \text{if } ||t_2 + \alpha - s_m|| < T \\ MC + ||\gamma_2|| & \text{if } ||t_2 - s_m|| > T \\ MC + ||\gamma_2|| & \text{No more maxima in model} \end{cases}$$

where T is a user-defined threshold.
3. Repeat 2 for all the elements in the target.
4. Calculate the CSS shift parameter using the second highest maximum (t_2, γ_2) in both the target and the model (s_2, σ_2) and also for maxima in the model that are close to the highest maximum of the target (within 80% of the maximum scale value). Repeat 1-3 for using these CSS shift parameters and calculate the match cost.
5. Repeat 1-4 by interchanging the place of the target and the model.
6. The lowest cost from all these matches is taken as the best match value.

A modification of this method described in [10.1] involves using global parameters of a shape such as eccentricity and circularity to eliminate certain shapes before the matching process. It also overcomes some of the problems of the original method like shallow concavities.

10.6.4 The SKS Model

The problem of making global decisions from a collection of local measurements is considered one of the fundamental problems of machine intelligence in general and computer vision in particular. This section provides a general approach [10.22] to the problem in the image analysis domain, describing a general methodology for recognizing shapes in images. The strategy begins by making local measurements at salient points in an image. The information from those local measurements is then fused using a strategy reminiscent of the Hough transform.

The method, called the *SKS* (Simple K-Space) algorithm, can be shown to be invariant to rotation in the camera plane, translation, and scale change, and very robust to partial occlusion and local variations in image brightness.

The algorithm was developed under the constraint that all operations must be reasonably computed by a biological neural network, characterized by:

- lots of memory
- simple computations
- low accuracy of those operations
- a great deal of parallelism.

The resulting algorithm uses accumulator arrays to search for consistency in shape recognition.

We begin with some definitions:

Object An object, to SKS, is anything in an image that we need to recognize. It could be the shape of a silhouette or the boundary of a region, if the application is to recognize objects by their silhouette. It could also be the neighborhood of an interest point, if the application is to find correspondences such as SIFT (see section 11.4) or a point on a boundary curve as described in this section. It could even be a complex scene made up of a number of features (e.g., a statue). In this chapter, we will describe the application to determining and matching the boundary of a region.

Feature point Every object is made up of one or many feature points. The boundary of a silhouette consists of points. Not every point is necessarily a feature point, and those that are not do not enter into the algorithm. A feature point in an image could be an interest point, to be described in section 11.5 or a point where the boundary changes sign.

Property list Each feature point has a list of properties. On the contour of a silhouette, a property might be the curvature of the contour at the feature point. In an image of a region, a property list might include the curvature and hue of the curve at that point. In an image analysis application, properties of the feature point might be the same as the features of the neighborhood. We use the symbol ν for the property list, written as a vector. The one exception to this is when the property list has a single element, usually curvature κ, and in that case the symbol for the scalar (e.g., κ) is used rather than ν.

Reference point The object being recognized usually has a single reference point.[9] The distance from the reference is denoted by the symbol ρ. The reference point is usually chosen to be the center of gravity of the region.

In SKS, the general philosophy is as follows:

Make a model. Construct a model that has the form

$$m(\rho, \nu) = \frac{1}{Z} \sum_{i=1}^{n} \delta(\rho - \rho_i)\delta(||\nu - \nu_i||), \tag{10.71}$$

where the Kronecker delta $\delta(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases}$. Such a model simply counts the number of times a point with property ν occurs at the distance ρ from the reference point. Suitably normalized, this count can approximate a probability. To allow for noise and errors in measurement, the model equation may be modified, allowing the delta functions to be replaced by Gaussians.

$$m(\rho, \nu) = \frac{1}{Z} \sum_{i=1}^{n} \exp\left(-\frac{(\rho - \rho_i)^2}{2\sigma_\rho^2}\right) \exp\left(-\frac{||\nu - \nu_i||^2}{2\sigma_\nu^2}\right) \tag{10.72}$$

where Z is the normalization, discussed later.

[9] If partial occlusion can occur, then multiple reference points may be used.

Use the model to match an image By scanning over the image, find the point that is most consistent with the model. That point will be the best match for the reference point of this model. This is accomplished by computing

$$A_C(\mathbf{x}) = \frac{1}{Z}\sum_k m(||\mathbf{x} - \mathbf{x}_k||, \mathbf{v_k}) \tag{10.73}$$

at every point $\mathbf{x}$, where $\mathbf{v_k}$ is a feature vector describing one point.

In this book, SKS appears in two forms. In this section, a model is developed for contours of regions, and in section 11.5, the same philosophy is used to build a descriptor for a local area around an interest point.

Modeling Contours with SKS

We consider a two-dimensional shape characterized by its outline. In fact, since the object being recognized is really just a 1D curve embedded in the plane, it is really a one-dimensional problem.

Let a shape be defined by a curve in the plane, $\alpha(s) = [x(s), y(s)]^{\mathrm{T}}$ parameterized by arc length, s. Then, we construct a function to be used as a shape model:

$$m(\rho, \kappa) = \int_s \delta(\rho - \rho(s))\delta(\kappa - \kappa(s))ds \tag{10.74}$$

where ρ denotes the Euclidean distance from the point s on the curve to a special reference point called Ω. $\Omega = [\Omega_x, \Omega_y]^{\mathrm{T}}$ is an arbitrary point in the plane. Once chosen, Ω must be held constant for the duration of the model building process.

$\kappa(s)$ in principal denotes any measurement or vector of measurements of the curve at point s. Curvature is suggested here, because it is invariant to translation and rotation.

In practice, the delta function, is approximated by a Gaussian,

$$\delta(\xi) \approx \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\xi^2}{2\sigma^2}\right), \tag{10.75}$$

which will provide additional averaging of noise.

Thus, a point a, b in the model is a measure of the likelihood that in this model, one may find a point that has curvature of b and is a distance a from the reference point.

In order to actually implement the integral of Eq. 10.74, the integral must, of course, be replaced by a summation, but this requires that one normalizes the result. Temporarily postponing actual definition of the normalization constant, we substitute Gaussians for the delta functions of Eq. 10.74 to find

$$m(\rho, \kappa) = \frac{1}{Z}\sum_i \frac{1}{\sigma_\rho}\exp\left(-\frac{(\rho - \rho_i)^2}{2\sigma_\rho^2}\right)\frac{1}{\sigma_\kappa}\exp\left(-\frac{(\kappa - \kappa_i)^2}{2\sigma_\kappa^2}\right) \tag{10.76}$$

where the normalization is Z for now.

One observation concerning Eq. 10.74 is relevant. The integral provides a count of the number of times a certain (ρ, κ) pair has been observed. It can be considered, therefore as an estimate of a probability–the probability that a particular value pair will be observed.

We have had good luck using the maximum rather than a sum in Eq. 10.76.

The definition of the model,

$$m(\rho, \kappa) = \frac{1}{Z} \sum \delta(|\rho - \rho_i|)\delta(|\kappa - \kappa_i|),$$

can be thought of as a count of the number of times a point is found on the curve c that is distance ρ from the origin and has local curvature κ. The set of all such sums, over possible values of ρ and κ is simply a histogram. One converts a histogram to an estimate of a probability density function by dividing by the total number of instances, which is really just the number of points along the curve.

This simple way of thinking of the model is slightly complicated by the replacement of the δ with the exponential, but the sense of the histogram idea still holds. When thinking about it that way, Z should be just n, the number of points on the curve. However, another normalization is required when matching a curve to an SKS model. For that reason, let us defer the normalization question for a few paragraphs.

Matching Contours with SKS Above, an algorithm was described that constructed a model for two-dimensional shapes. That model was referred to as m, $m : U \times \mathbb{R}^d \rightarrow \mathbb{R}$, where $U = (0, maxdist)$, and it was a function of two parameters, where the first is a distance and the second a d-dimensional feature vector,

To use this model in matching an unknown discrete shape, say C to a model m_j, we compute the function

$$A_C(\mathbf{x}) = \frac{1}{Z} \sum_k m(||\mathbf{x} - \mathbf{x}_k||, \mathbf{v_k}) \tag{10.77}$$

where $\mathbf{x}$ is a point in the image plane, $\mathbf{x}_k$ is the kth point on the curve C, and $\mathbf{v}_k$ is the feature vector at that k-th point. If $\mathbf{v}_k$ is a scalar, curvature is a good choice.

The function A will be steeply peaked and (in the limit of a perfect match) be equal to 1.0 at the reference point chosen in developing M.

Figure 10.19 illustrates the difference between matching a shape to itself and to another, similar, object in the same class (tanks). With appropriate normalization, the maximum value for $A(\mathbf{x})$ can be forced to be 1.0 and located at the correct value of $\mathbf{x}$. Then, classifications reduce to a simple threshold.

The normalization is accomplished as follows: A model is matched with itself, and the quality of the match is recorded. Since matching with itself is the best possible value, that value is used as the normalization of the model.

In Figure 10.20 an outline is shown of a pistol partially occluded by a hammer. SKS can identify both objects from this image. A neural network is able to solve the same problem with comparable performance [10.16]. Experimentally, on the set of fish silhouettes from which Figure 10.18 was drawn, the silhouette matching version of SKS performed somewhat better than CSS or Shape Context.

10.7 Geodesics in a Shape Space

A manifold is a set of points interpreted as a surface in which each point has a local neighborhood that is Euclidean. That is, the geometry you learned in high school applies on a

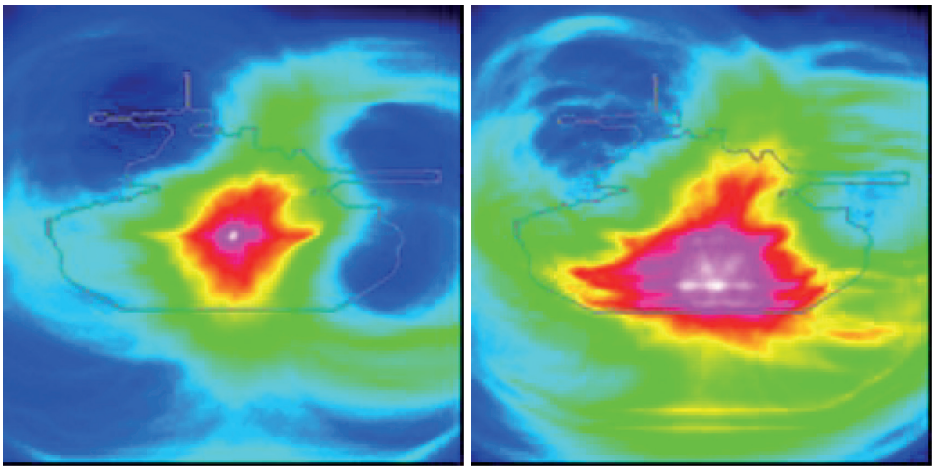


Figure 10.19 The pseudo-colored image on the left shows the accumulator array for a good match (actually, matching one image to a rotated and scaled version of itself). The bright spot defines the estimated location of the reference point, and is sharply peaked. The image on the right shows the result of matching images of two different but similar tanks. The bright spot/area is diffuse.

manifold, but only locally. For example, Euclid [10.27] stated that two parallel lines would never intersect. This is true on a plane, but not on the surface of a sphere. Lines of longitude on the surface of the earth are parallel at the equator but intersect at the poles. Similarly, triangles on the surface of a sphere do not have exactly 180º of internal angles (Figure 10.23). Thus, a geometry constrained to the surface of a sphere must be non-Euclidean. The concept of a non-Euclidean geometry leads to thinking about manifolds, because manifolds present a good way to think about general surfaces.

Manifolds allow elegant representations of curves and surfaces, and familiarity with the basic concepts of projection onto a manifold and finding the shortest path between two points on a manifold (a geodesic) are helpful to the Computer Vision student.

We once again think about matching two shapes, but we think of shapes in a different way, as a single point in a very high-dimensional space. Furthermore, the distance between the two points will be defined in a different way – it will be the distance measured along a particular path called a *geodesic*. The concept of a manifold is fundamental to shape description, but does require a higher level of abstraction than many of the other sections in this book.

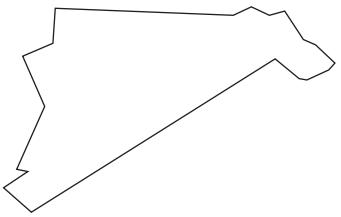


Figure 10.20 Silhouette of a gun partially occluded by a hammer. Redrawn from[10.16]

10.7.1 A Shape in Two Dimensions

A shape may be thought of as a property of the boundary of a region in an image. For that reason, we will often interchangeably use the words "contour," "boundary," and "curve." A shape in 2D can be considered a vector-valued function of arc length, s. Thus, in the familiar Cartesian coordinates, a curve in the plane may be written as the vector function $\boldsymbol{\alpha}(s) = \begin{bmatrix} x(s) \\ y(s) \end{bmatrix}$ where $0 \leq s < 2\pi$. Limiting s to lie between 0 and a constant value such as 2π forces the perimeter of any shape to be the same, thus providing invariance to scale changes. The concepts we are about to introduce are not restricted to curves in the plane, as we can easily write $\boldsymbol{\alpha}(s) = [x(s)\ \ y(s)\ \ z(s)]^{\mathrm{T}}$, for a curve in 3-space (think of a straightened coathanger).

Using s as the parameter of the curve is convenient because at every point, the first derivative of $\boldsymbol{\alpha}$ is the tangent to the curve at that point.

$$\boldsymbol{T} = \frac{d\boldsymbol{\alpha}}{ds}, \tag{10.78}$$

and the second derivative is the *normal*, which is normally[10] defined as a unit vector:

$$\boldsymbol{N} = \frac{\frac{d\boldsymbol{T}}{ds}}{||\frac{d\boldsymbol{T}}{ds}||}. \tag{10.79}$$

Finally, the cross product of those two is the *binormal*,

$$\boldsymbol{B} = \boldsymbol{T} \times \boldsymbol{N}. \tag{10.80}$$

Since all three vectors, Tangent, Normal, and Binormal, are mutually orthogonal, they define a particular coordinate frame, the *Frenet frame*. The three vectors and their derivatives are also related proportionally, by

$$\frac{d\boldsymbol{T}}{ds} = \kappa \boldsymbol{N} \tag{10.81}$$

$$\frac{d\boldsymbol{N}}{ds} = -\kappa \boldsymbol{T} + \tau \boldsymbol{B} \tag{10.82}$$

$$\frac{d\boldsymbol{B}}{ds} = -\tau \boldsymbol{N}, \tag{10.83}$$

where κ is the *curvature* (yes, this is the same curvature we referred to earlier), and τ is the *torsion*. You can think of the torsion as the twisting of the Frenet frame when it follows a nonplanar curve.

Other representations for curves may also be used: For example, as we discussed in section 10.4.4, instead of an ordered list of coordinate pairs, we could consider each point as a complex number, $x + iy$, which allows us to think of the points as scalars. Or, we could consider the direction of the tangent - the angle the tangent makes with the x axis, or the direction the normal makes. The tangent angle, $\theta_n(s)$ is particularly convenient, and is used

[10] Yes, that pun is deliberate.

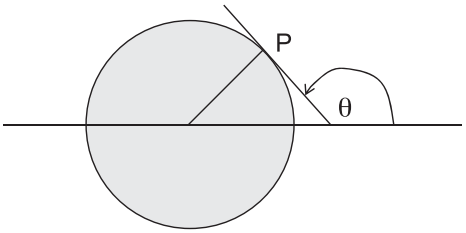


Figure 10.21 On the unit circle, the direction of the tangent to a curve at a point P is a simple scalar θ.

here, because it is easy to visualize, and because on the unit circle, that direction is related to the arc length, by nothing more than addition of a constant,

$$\begin{cases} \theta_n(s) = s + \frac{\pi}{2} & \text{for } 0 <= s < \frac{3\pi}{2} \\ \theta_n(s) = s - \frac{3\pi}{2} & \text{for } \frac{3\pi}{2} <= s < 2\pi \end{cases},$$

as is illustrated in Figure 10.21. Of course the unit circle isn't a very interesting shape. But, observe that the average value of θ, defined by $\frac{1}{2\pi}\int_0^{2\pi}\theta(s)ds = \pi$.

In most of the remainder of this section the tangent direction form, either $\theta(s)$ (continuous) or $[\theta_1, \theta_2, \ldots, \theta_n]^{\mathrm{T}}$ (discrete) will be used, since each element of the vector is a real scalar.

The requirement that the average value of the function is equal to π will come up again.

Another feature we could parameterize with s and use for describing a curve is curvature. We will not describe that approach here, but you should learn this one important theorem: *Any regular curve with non-zero curvature has its shape (and size) completely determined by its curvature and torsion.* This is the fundamental theorem of curves and might come in useful someday.

10.7.2 A Closed Boundary as a Vector

A sampled boundary with, say, 100 samples, may be written as a 100-dimensional vector. Figure 10.22 illustrates a boundary with nine points. So with 100 points, we have a 100-dimensional vector. No problem. A thousand points? Still no problem. What if we consider the boundary as continuous instead of sampled? It is perfectly OK to continue to think of it as a vector, but as s ranges from 0 to 2π, it takes on an infinite number of different values and the representation of the curve becomes an infinite-dimensional vector. We call those "functions." We can't write it out as a table, but we probably didn't want to write out the 1000-dimensional one either. As we did in Chapter 3, we simply use the functional notation.

But there is more than one infinity that confronts us. Not only is each contour represented by an infinite-dimensional vector, there could be an infinity of them. Think of a single curve, like the one illustrated in Figure 10.22. You could rotate that curve about some point of interest (e.g., its center of gravity), and in so doing create an infinity of new curves, one for each possible rotation.

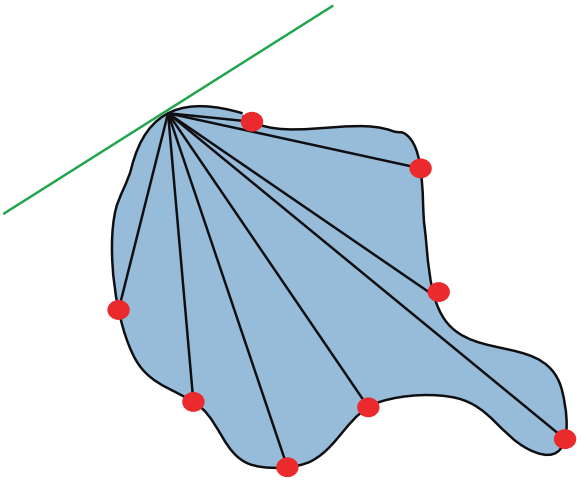


Figure 10.22 A boundary with only nine points. The points may be listed as ordered pairs, using Cartesian (x, y), or polar (ρ, θ). If the sample points are equally spaced in arc length (they are not, in this drawing), following the idea of "go one step (of distance 1.0), turn through angle θ_i, and continue." Following this convention, the list of coordinates is simply a list of the turning angles, scalars.

As discussed in section 10.2, rotation in 2D is accomplished by multiplying each point by a rotation matrix,

$$\begin{bmatrix} x'(s) \\ y'(s) \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x(s) \\ y(s) \end{bmatrix}$$

Rotation matrices like this have the property that their columns are orthonormal vectors, and they are referred to as "orthonormal transformations." We refer to the collection of such rotations as the "Special Orthogonal group," $SO(n)$, where n refers to the dimension of the operator. In the cases described in this section, we consider $n = 2$.

There is another group of possible operations that can be applied to a boundary without changing the shape. This involves moving the starting point and is called the "reparameterization group," $\mathcal{D}$. The reparameterization group also includes the possibility that the parameter may vary in its speed as it moves around the curve, but we are going to ignore that possibility here.

Now we have, for any single shape, an infinity of possible versions, all of which are in some sense equivalent. Since we are using the concept of vectors to represent each shape, and since the conditions of section 3.2.2 are easily shown to be true, we have defined a vector space.

10.7.3 Vector Spaces

Remember from section 3.2.2 that a vector space is a collection of vectors that satisfy the requirement that if two vectors are members of the same space, their sum is likewise in the same collection. Furthermore, if ω is a scalar, and V an element of the collection of vectors, then ωV is also a member of the collection. To be really complete in the definition,

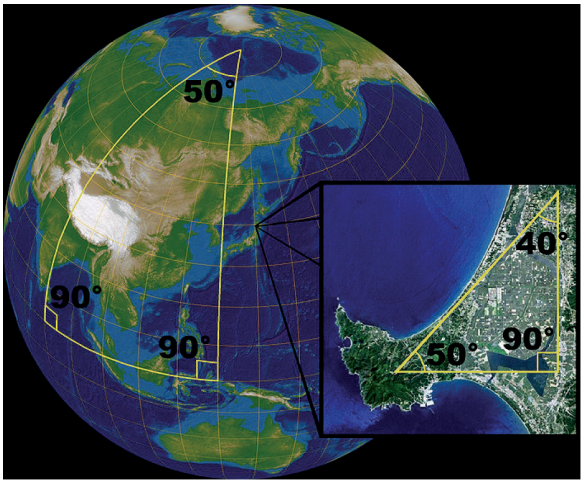


Figure 10.23 A triangle drawn on a sphere does not have 180 degrees, so a sphere is not globally Euclidean, but it is locally Euclidean. Figure from `https://en.wikipedia.org/wiki/Spherical_geometry`.

we have to specify the properties of vector addition (associative, distributive, etc.), but let's not bother with that right now.

Since we can write a shape as a vector, and we can add and multiply those vectors to produce new vectors that we can interpret as shapes, a single shape is an element of a vector space.

There are two special vector spaces that are important here: The first is the space of all square-integrable functions, $\mathbb{L}^2$, that is, all functions f such that $\int_a^b f^2(t)dt < \infty$. The second is the space of all possible closed shapes, that we will call $\mathcal{C}$. In a later section, we will present the properties that define $\mathcal{C}$.

10.7.4 Manifolds

The special space, $\mathcal{C}$ defined above, has the property that it is locally Euclidean. To see what this means, think about the surface of a sphere. The sum of the interior angles of a triangle does not equal 180 degrees when the triangle is drawn on a sphere (see Figure 10.23), Yet, if the sphere is big enough, or the triangle small enough, the difference between the sum of the interior angles and 180 degrees becomes vanishingly small. Furthermore, the process through which it becomes vanishingly small is smooth. A sphere thus has a "locally Euclidean" geometry. Any surface with such a property is called a "manifold." We thus refer to $\mathcal{C}$ as the manifold of possible closed curves, and will not attempt to prove the Euclidean nature of the manifold here.

In the example of the surface of the earth, the manifold is a 2-manifold, that is, any point is uniquely identified by two parameters, longitude and latitude. Furthermore, there are an infinite number of points in that manifold. It is continuous and differentiable. With just two parameters, we have no way to talk about points that are inside the earth, but when it becomes necessary to use such terminology, we talk about a manifold "embedded in" a

higher dimensional space. For example, we think of the surface of the earth as embedded in a 3-space.

A *manifold* is merely a set of points with some continuity constraints. Most of the time, we think of it as a continuous surface, but it is really no big deal.

It will often be convenient to use terminology describing the manifold as a surface, so you will see terms like "on the manifold."

Recall in section 8.4.2, we talked about *equivalence relations*? Here, we define a different equivalence relation as follows: two vectors v_1 and v_2 are equivalent if they both represent the same shape, independent of rotation and parameterization. With this definition, $\mathcal{C}$ is partitioned into equivalence classes, one for each distinct shape. Since there are an infinite number of possible 2D closed shapes with a particular perimeter, the number of equivalence classes is still infinite. All of the different elements of $\mathcal{C}$ corresponding to a particular shape (that is, all the elements of one particular equivalence class) defines an *orbit*. You may also see the set of equivalence classes referred to as the "quotient set."

Let I denote the interval $I = [0, 2\pi]$, and let $\beta : I \to \mathbb{R}^2$ be a curve in the plane. Then, if $s \in I$ is arc length we could describe curves not only in terms of their coordinates, but in terms of their directions, $\theta(s)$ as described above, or by their velocity vectors,

$$q(s) = \frac{\dot{\beta}(s)}{\sqrt{||\dot{\beta}(s)||}}. \tag{10.85}$$

We observe that this function, evaluated at any particular s, is tangent to β. Furthermore, $\frac{q(s)}{||q(s)||}$ is the unit tangent vector at s.

10.7.5 Projecting onto the Manifold of Closed Curves

In this subsection, an illustration of the philosophy, the approach presented in [10.18] is followed. We will describe curves as continuous functions and also as very long vectors, and we will discuss (as we did earlier) how to go back and forth between the representations.

Two ways present themselves for representing curves.

In a function space A single function, $\theta(s)$, where s is the usual arc length and θ is the angle the tangent to the curve makes with the x axis.

In a discrete vector space A curve is a list of all the points around the curve. In this discrete form, s is uniformly sampled, and so $\theta(s_i)$ denotes the operation of traveling a distance Δs in the direction θ. Now instead of a continuous function, the curve is a high-dimensionality (e.g., 256 elements, not infinity) vector.

When thinking of a curve as a function, we denote the curve by $\theta(s)$, where θ is the direction of the tangent at point s.

When thinking of a curve as a vector, we denote it by a boldface symbol, such as $\boldsymbol{\theta}$. Since these vectors have an infinite number of elements (until we quantify them), the two representations are really the same.

We begin by using the continuous form.

We ask that you take the following three equations on faith until you take another math course. Any curve, $\theta(s)$ must satisfy the following three equations if the curve is closed:

$$\begin{aligned}\Phi_1(\theta) &\equiv \frac{1}{2\pi}\int_0^{2\pi}\theta(s)ds = \pi \\ \Phi_2(\theta) &\equiv \int_0^{2\pi}\sin\theta(s)ds = 0 \\ \Phi_3(\theta) &\equiv \int_0^{2\pi}\cos\theta(s)ds = 0.\end{aligned} \tag{10.86}$$

We begin to think now of the set of closed curves in the plane as being a manifold, and begin to use the term "the manifold," where we really mean the manifold of closed curves in the plane.

Suppose we have two closed curves $a(s)$ and $b(s)$. If we think of them as vectors, then by definition they lie on the manifold. Each of those curves consists of an infinite number of points, and we can still think of it as an ordered list of scalars, a vector. A straight line could be drawn between any two such points. Question: is every intermediate point also a closed curve? To find points on a straight line between two points[11] is pretty easy, as follows:

Call the first vector A and the second vector B. Assuming s was fairly finely sampled, these vectors could have dimensions around a thousand or so. The point

$$C = \alpha A + (1-\alpha)B \tag{10.87}$$

is a point on the straight line (a Euclidean straight line) between A and B if $0 \leq \alpha \leq 1$. So just let α range from 0 to 1 and you will generate intermediate vectors that lie on the straight line between A and B. The lower set of curves in Figure 10.24 illustrates this, and clearly, the intermediate points are not closed curves. Given one of these intermediate points, say $\boldsymbol{\theta}$, we need to find the projection of that point onto the manifold. Here, "projection" will simply mean finding the point on the manifold closest to the point of interest. We approach this by finding a small step in the direction of the manifold. Let that step be named $h(s)$, and we add it to $\theta(s)$ at each value of s.

$$\Phi_1(\theta+h) = \frac{1}{2\pi}\int_0^{2\pi}(\theta(s)+h(s))ds = \Phi_1 + \frac{1}{2\pi}\int_0^{2\pi}h(s)ds \tag{10.88}$$

$$\Phi_2(\theta+h) = \int_0^{2\pi}(\cos(\theta(s)+h(s))ds \tag{10.89}$$

$$\Phi_3(\theta+h) = \int_0^{2\pi}(\sin(\theta(s)+h(s))ds. \tag{10.90}$$

[11] Since a vector of dimension d represents a single point in a d-dimensional space, we use the words *vector* and *point* interchangeably here.

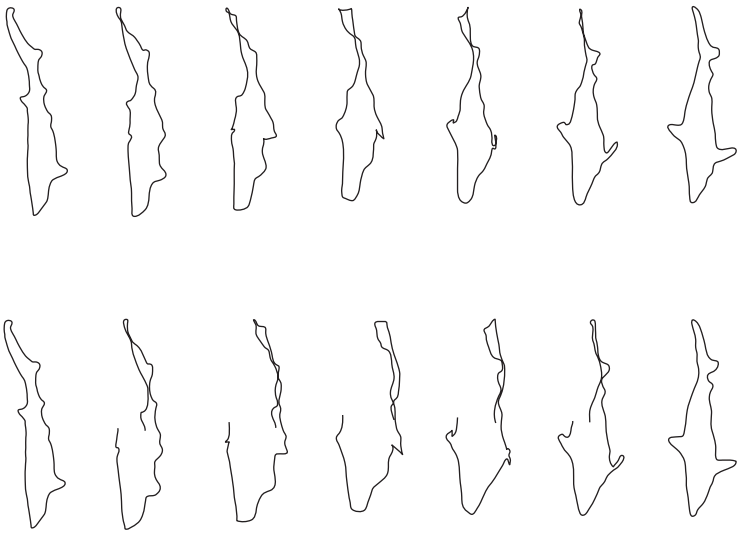


Figure 10.24 Moving left to right, from a point on the manifold that represents one type of shark to another. The upper set of curves are vectors, all of which are part of a geodesic on the manifold. The lower set of curves moves in a straight line through the 256-dimensional Euclidean space. Clearly, the second set of vectors are not all closed curves.

With Eqs. 10.89 and 10.90 one can do a bit of algebra involving the sine and cosine of a sum and the properties of the sine and cosine of small numbers to find that

$$\begin{aligned}
\Phi_1(\theta + h) &= \Phi_1 + \frac{1}{2\pi}\int_0^{2\pi} h(s)ds \\
\Phi_2(\theta + h) &= \Phi_2 - \int_0^{2\pi} \sin(\theta(s))h(s)ds \\
\Phi_3(\theta + h) &= \Phi_3 + \int_0^{2\pi} \cos(\theta(s))h(s)ds.
\end{aligned} \tag{10.91}$$

Defining the small movements as

$$\begin{aligned}
d\Phi_1 &\equiv \frac{1}{2\pi}\int_0^{2\pi} h(s)ds \\
d\Phi_2 &\equiv -\int_0^{2\pi} \sin(\theta(s))h(s)ds \\
d\Phi_3 &\equiv \int_0^{2\pi} \cos(\theta(s))h(s)ds.
\end{aligned} \tag{10.92}$$

At this point, let's switch from vectors of infinite length in a functional space to very long vectors with d elements, where d might be a thousand. There isn't much practical difference, but it will allow a more convenient notation.

Observe that $d\Phi_1$ can be written as an inner product:

$$d\Phi_1 = \left\langle \frac{1}{2\pi}, \mathbf{h} \right\rangle, \tag{10.93}$$

where $\mathbf{h} = [h_1, h_2, \ldots, h_d]^{\mathrm{T}}$, and similarly

$$d\Phi_2 = \langle -\sin(\theta), \mathbf{h} \rangle$$
$$d\Phi_3 = \langle \cos(\theta), \mathbf{h} \rangle .$$

These three equations can be written as a matrix-vector product, as

$$d\Phi(\mathbf{h}) = J\mathbf{h} \tag{10.94}$$

where the Jacobian matrix J is found by expanding the three inner products:

$$J = \begin{bmatrix} \frac{1}{2\pi} & \frac{1}{2\pi} & \cdots & \frac{1}{2\pi} \\ -\sin(\theta_1) & -\sin(\theta_2) & \ldots & -\sin(\theta_d) \\ -\cos(\theta_1) & -\cos(\theta_2) & \ldots & -\cos(\theta_d) \end{bmatrix}. \tag{10.95}$$

Before we get too lost in the math, let's remember where we are and what we are doing. We are at a point $\boldsymbol{\theta}$ in the d-dimensional space. At this point, we can evaluate the vector Φ, and if that is not equal to $[\pi, 0, 0]^{\mathrm{T}}$, we are not on the manifold. We take a small step, by adding $\mathbf{h}$ to $\boldsymbol{\theta}$, (remember, $\mathbf{h}$ is a vector!), and we would like that step to take us closer to the manifold; but we don't know what small step, $\mathbf{h}$, would do that.

Define the *residual*, to be the difference between where we want to be and where we are, to be $\mathbf{r}$,

$$\mathbf{r} = \begin{bmatrix} \pi \\ 0 \\ 0 \end{bmatrix} - \Phi \tag{10.96}$$

and try to find a value of $\mathbf{h}$ for which $J\mathbf{h}$ will be close to $\mathbf{r}$. In addition, we want the smallest $\mathbf{h}$ that will do the job.

Like we have seen before, simply minimizing $J\mathbf{h} - \mathbf{r}$ won't work because it can be minimized by useless things like $\mathbf{h} = 0$. Therefore let's insist on making $\mathbf{h}$ satisfy $J\mathbf{h} - \mathbf{r} = 0$, and then define a Lagrangian:

$$L = \frac{1}{2}\mathbf{h}^{\mathrm{T}}\mathbf{h} + \Lambda^{\mathrm{T}}(J\mathbf{h} - \mathbf{r}) = \frac{1}{2}\mathbf{h}^{\mathrm{T}}\mathbf{h} + \Lambda^{\mathrm{T}}J\mathbf{h} - \Lambda^{\mathrm{T}}\mathbf{r}. \tag{10.97}$$

By minimizing this for $\mathbf{h}$, we will have a step that will take us at least close to the manifold. Differentiate with respect to $\mathbf{h}$, and set to zero:

$$\mathbf{h} = -J^{\mathrm{T}}\Lambda. \tag{10.98}$$

Now we have two equations involving J and $\mathbf{h}$:

$$J\mathbf{h} = \mathbf{r} \tag{10.99}$$
$$\mathbf{h} = -J^{\mathrm{T}}\Lambda. \tag{10.100}$$

Solving this for $\mathbf{h}$ looks pretty easy: just use Eq. 10.99 and solve it for $\mathbf{h}$ using the pseudoinverse (recall section 12.6.4) as follows:

$$J\mathbf{h} = \boldsymbol{r}$$
$$J^{\mathrm{T}}J\mathbf{h} = J^{\mathrm{T}}\boldsymbol{r}$$
$$\mathbf{h} = (J^{\mathrm{T}}J)^{-1}J^{\mathrm{T}}\boldsymbol{r},$$

and we are done! That was easy, wasn't it?.... uh .... wasn't it?

Wait: J has 3 rows and something like 1000 columns. So $(J^{\mathrm{T}}J)$ is something like 1000×1000, a matrix that is computable, but probably not invertible.

What we need to do is to manipulate the algebra until we find something involving $(JJ^{\mathrm{T}})^{-1}$, which is 3×3 and not $(J^{\mathrm{T}}J)^{-1}$. Start with solving Eq. 10.100 for Λ. Again, use the pseudoinverse and find

$$\Lambda = -(JJ^{\mathrm{T}})^{-1}J\mathbf{h}. \tag{10.101}$$

Now, multiply both sides of this equation by J^{T}:

$$-J^{\mathrm{T}}\Lambda = J^{\mathrm{T}}(JJ^{\mathrm{T}})^{-1}J\mathbf{h}. \tag{10.102}$$

Observe that from Eq. 10.100, the left-hand side is just $\mathbf{h}$, and the last two terms on the right-hand side are just $\boldsymbol{r}$, and we get our final, computable answer:

$$\mathbf{h} = J^{\mathrm{T}}(JJ^{\mathrm{T}})^{-1}\mathbf{r}. \tag{10.103}$$

Thus, given a point, $\boldsymbol{\theta}$ in d space, we can find the projection of that point onto the manifold of close shapes by

1. Compute $\mathbf{r}$ and J.
2. Compute $\mathbf{h}$.
3. Add $\mathbf{h}$ to $\boldsymbol{\theta}$.
4. Iterate. Yes, iterate. This derivation started out using first order Taylor series, so it only gets us close to the solution. This projection algorithm normally converges to a useful solution in just 2 or 3 iterations.

   Now we have a method that will take us from a point in d-space to a point on the manifold, but that isn't what we wanted. We wanted the geodesic between the two points.

10.7.6 Finding a Geodesic

A geodesic between two points on a surface is a path on the surface that is of minimum length. On the plane, geodesics are unique, however on other surfaces, they may not be unique. For example, on the earth, there is an infinite number of minimum length paths from the north pole to the south. Interestingly, every geodesic on the surface of a sphere is part of a circle (called a *great circle*) whose center is at the center of the earth.

If we can find a set of points that go from the first point to the second, remaining on the manifold, and we have a way to minimize the length of that path, we have a geodesic[12]. That is, the geodesic is the shortest curve on the manifold.

Assume we have two points, $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_f$ (initial and final), both of which are known to lie on the manifold, and we seek the geodesic between them.

Begin by defining an operator $P(\boldsymbol{\theta})$ that projects $\boldsymbol{\theta}$ onto the manifold of closed shapes. This is the projection operator we just derived.

1. Using Eq. 10.87, find n equally spaced points, $\mathbf{d}_i$ along the Euclidean line between $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_f$, including $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_f$, as illustrated in Figure 10.25, and name these points $\mathbf{d}_1 - \mathbf{d}_n$.

[12] The authors are grateful to Lyle Noakes for his help in deriving this formulation.

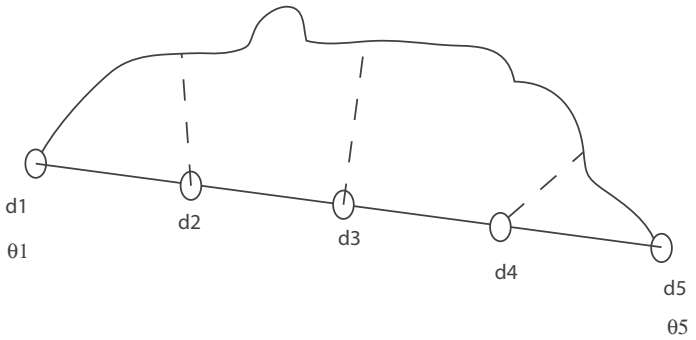


Figure 10.25 n points are found that lie along a straight line between $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_f$. Each of those is then projected onto the manifold to find new value for the $\boldsymbol{\theta}$'s.

Find the projections of all these points onto the manifold, and denote them $\boldsymbol{\theta}_1 - \boldsymbol{\theta}_n$. Since $\boldsymbol{\theta}_i = \boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_f = \boldsymbol{\theta}_n$ are already on the manifold, there is no need to project them.

2. Compute new values for the d's, using, as shown in Figure 10.26

$$\mathbf{d}_i = (\mathbf{d}_{i-1} + \boldsymbol{\theta}_{i+1})/2, \quad i = 2, \ldots, n-1. \tag{10.104}$$

This process, estimating $\mathbf{d}_k$ from $\mathbf{d}_k - 1$ and $\boldsymbol{\theta}_{k+1}$, is called *leapfrog*. The set of points defined by the $\mathbf{d}_i$ defines a *polyline*, an end-to-end set of straight line segments. Resample that polyline into n equally spaced (along the polyline) segments.

These will be new estimates of the points $\mathbf{d}_i$.

3. Project those d's as above and iterate until converged. Thus usually requires just a couple of iterations.

After convergence, each of these points lies on a geodesic of the manifold of close shapes. This process is illustrated in Figure 10.24. In that figure, the top line illustrates one type of shark at the far left, and a different type at the far right. Both are closed curves. By using the method described in this section, intermediate values may be computed that are also closed curves, unlike the lower portion of Figure 10.24.

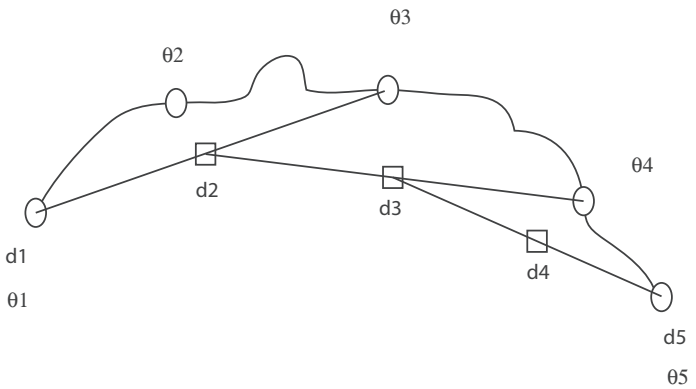


Figure 10.26 New values for the d's are computed, and these will lie closer to the manifold. For example, $\mathbf{d}_3 = (\mathbf{d}_2 + \boldsymbol{\theta}_4)/2$.

10.8 Conclusion

In this chapter, several features were defined that could be used to quantify the shape of a region. Some, like the moments, are easy measurements to make. Others, such as the diameter or the convex discrepancy, require development of fairly sophisticated algorithms to avoid long computation time. There have also been some efforts in automatic learning of shape descriptors [10.40].

Invariance to various linear transformations is essential in the design of robust shape descriptors. Studies of signal processing in the mammalian visual cortex [10.32] have suggested that images are represented in the cortex after being subjected to a log-polar transformation, and that invariance may be provided in this way [10.31][10.39]. Other transformations [10.37] may provide equivalent or superior representations in computer applications.

One topic of significant interest that is not covered in this chapter is how a shape may change in time. For example, consider the analysis of the gait of a walking individual. This requires not only shape analysis, but analysis of how the shape is changing. A paper by Veeraraghaven et al. [10.38] presents this topic clearly, and thoroughly.

Optimization methods came up quite a few times in this chapter:

- In section 10.3.2, a derivation is presented that finds the straight line that best fits a set of points, in the sense that the sum of perpendicular distances is minimized. To accomplish that, we were required to use constrained minimization with Lagrange multipliers.
- Constrained optimization was again used to find the best basis vectors when doing principal components.
- The shape context requires a solution to the Linear Assignment Problem.
- Both CSS and SKS search over relatively small search spaces for maxima.

10.9 Assignments

Assignment 10.1: Show that $\mathbf{x}' = R\mathbf{x} + \mathbf{b}$ is not a linear operation for a rotation R and translation vector $\mathbf{b}$.

Assignment 10.2: For each feature described in section 10.4.1, determine if that feature is invariant to (1) rotation in the viewing plane, (2) translation in that plane, (3) rotation out of that plane (an affine transform if the object is planar), and (4) zoom.

Assignment 10.3: Let the Euclidean distance between two points be denoted by the operator $d(P_1, P_2)$. (You may want to use this in the next problem.) Design a monotonic measure $R(P_1, P_2)$ that maps all distances to be between 0 and 1. That is, if $d(P_1, P_2) = \infty$, then $R(P_1, P_2) = 1$, and if $d(P_1, P_2) = 0$, then $R(P_1, P_2) = 0$.

For the metric you developed, show how you would prove your measure is a formal metric. Just set up the problem. Extra credit if you actually do the proof.

Assignment 10.4: Following are five points on the boundary of a region: (1,1), (2,1), (2,2), (2,4), (3,2). Use eigenvector methods to fit a straight line to this set of points, thus finding

the principal axes of the region. Having found the principal axes, then estimate the aspect ratio of the region.

Assignment 10.5: In the online image database is an image named `blob.ifs` with black foreground and white background.

(1) compute the first three invariant moments of the foreground region.
(2) rotate the foreground region about its center of gravity through 10, 20, and 40 degrees, and compute the invariant moments of the resulting image. Include your rotated images with your homework. What do you conclude?

Assignment 10.6: Discuss the following postulate: Let P_1 and P_2 be two points in a region that determine the diameter of the region. Then P_1 and P_2 are on the boundary of the region.

Assignment 10.7: Starting with the first part of Eq. 10.31, prove the second part.

Assignment 10.8: What is the difference between the intensity axis of symmetry and the medial axis?

Assignment 10.9: Prove Eq. 10.56.

Assignment 10.10: In Table 10.1, prove that the invariant moment, ϕ_1 is invariant to rotation.

Assignment 10.11: Your instructor will specify an image containing a single region with unity brightness and zero background.

(1) Compute the seven invariant moments of the foreground region.
(2) Rotate the foreground region about its center of gravity through ten, twenty and forty degrees, and compute the invariant moments of the resulting image. What do you conclude?

Assignment 10.12: Prove that Eq. 10.49 is invariant to (a) translation, (b) rotation, and (c) zoom.

Assignment 10.13: Is the caption on Figure 10.11 correct?

Assignment 10.14: Two silhouettes, A and B, are measured and their boundaries encoded. Then, Fourier descriptors are computed. The descriptors are listed in Table 10.3. Is it possible that these two objects represent isometries of one another?

Could they be similarity transforms of one another?

Table 10.3 Complex values of the Fourier descriptor

object A	object B
5.00 + i 0.00	5.83 + i 0.0
4.2 + i 2.0	3.69 + i 3.15
3.86 + i 1.00	3.48 + i 2.00
2.95 + i 2.05	2.30 + i 2.77
3.19 + i 1.47	2.70 + i 2.24

Could they be affine transformations of one another? (An affine transformation is a linear transformation that includes not only rigid body motion, but also the possibility for scaling of the coordinate axes. If both axes [in 2D] are scaled by the same amount, you get zoom. If they are scaled by different amounts, you get shear.)

If you decide that these two sets of descriptors represent the same shape, possibly transformed, describe and justify what type of operations convert A into B. If they are not the same shape, explain why.

Assignment 10.15: A cylinder with unit radius and height of ten is oriented vertically about the origin, and is known to have a surface that is a Lambertian reflector. That is, the reflected brightness is independent of the angle of observation, and depends on the angle of incidence following the relationship: $f = aI\cos\theta_i$ where a is the albedo, I is the brightness of the source. On this cylinder, the albedo is constant.

The camera is located at $x = 0, y = -2, z = 0$, and the optical axis of the camera is horizontal and pointed at the origin.

The light source is known to be 4 units from the origin, and is known to be a point source that radiates equally in all directions. The brightest spot in the image of the cylinder is at an angle of 29 degrees, and that angle is measured in the horizontal plane from the viewing axis of the camera.

Where is the light source?

Bibliography

[10.1] S. Abbasi, F. Mokhtarian, and J. Kittler. Curvature scale space image in shape similarity retrieval. *Multimedia Syst.*, 7(6), 1999.

[10.2] V. Anh, J. Shi, and H. Tsai. Scaling theorems for zero crossings of bandlimited signals. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(3), 1996.

[10.3] H. Arbter, W. Snyder, H. Burkhardt, and G. Hirzinger. Application of affine-invariant fourier descriptors to recognition of 3-d objects. *IEEE Trans. Pattern Anal. and Machine Intel.*, 12(7), July 1990.

[10.4] S. Belongie and J. Malik. Matching with shape context. In *IEEE Workshop on Content-based Access of Image and Video Libraries (CBAIVL-2000)*, 2000.

[10.5] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. In *Technical Report UCB//CSD00 -1128*. UC Berkeley, January 2001.

[10.6] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. and Machine Intel.*, 24(4), April 2002.

[10.7] A. Califano and R. Mohan. Multidimensional indexing for recognizing visual shapes. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(4), April 1994.

[10.8] J. Chuang, C. Tsai, and M. Ko. Skeletonization of three-dimensional object using generalized potential field. *IEEE Trans. Pattern Anal. and Machine Intel.*, 22(11), November 2000.

[10.9] G. Dantzig. *Origins of the simplex method A history of scientific computing*. Systems Optimization Laboratory, Stanford University, 1987.

[10.10] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley Interscience, 2nd edition, 2000.

[10.11] A. Ferreira and S. Ubeda. Computing the medial axis transform in parallel with eight scan operations. *IEEE Trans. Pattern Anal. and Machine Intel.*, 21(3), March 1999.

[10.12] R. Gonzalez and P. Wintz. *Digital Image Processing*. Pearson, 1977.
[10.13] M. Gruber and K. Hsu. Moment-based image normalization with high noise-tolerance. *IEEE Trans. Pattern Anal. and Machine Intel.*, 19(2), February 1997.
[10.14] D. Helman and J. JáJá. Efficient image processing algorithms on the scan line array processor. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(1), January 1995.
[10.15] M. Hu. Visual pattern recognition by moment invariants. *IRE Trans. Information Theory*, 8, 1962.
[10.16] K. Sohn, J. Kim, and S. Yoon. A robust boundary-based object recognition in occlusion environment by hybrid hopfield neural networks. *Pattern Recognition*, 29(12), December 1996.
[10.17] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38, 1987.
[10.18] S. Joshi, E. Klassen, A. Srivastava, and I. Jermyn. A novel representation for Riemannian analysis of elastic curves in Rn. In *Computer Vision and Pattern Recognition, CVPR07*, June 2007.
[10.19] K. Kanatani. Comments on symmetry as a continuous feature. *IEEE Trans. Pattern Anal. and Machine Intel.*, 19(3), March 1997.
[10.20] H. Kauppinen, T. Seppnen, and M. Pietikinen. An experimental comparison of autoregressive and Fourier-based descriptors in 2d shape classification. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(2), February 1995.
[10.21] D. G. Kendall, D. Barden, T. K. Carne, and H. Le. *Shape and Shape Theory*. Wiley, 1999.
[10.22] K. Krish, S. Heinrich, W. Snyder, H. Cakir, and S. Khorram. A new feature based image registration algorithm. In *ASPRS 2008 Annual Conference*, April 2008.
[10.23] Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2, 1955.
[10.24] S. Liao and M. Pawlak. On image analysis by moments. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(3), March 1996.
[10.25] F Mokhtarian, S Abbasi, and J Kittler. Efficient and robust retrieval by shape through curvature scale space. In *Proceedings of the First International Workshop on Image Databases and Multi-Media Search*, pages 35–42, August 1996.
[10.26] T. Nguyen and B. Oommen. Moment-preserving piecewise linear approximations of signals and images. *IEEE Trans. Pattern Anal. and Machine Intel.*, 19(1), January 1997.
[10.27] Euclid of Alexandria. *Elements*. Unknown, 300 BC.
[10.28] L. O'Gorman. Subpixel precision of straight-edged shapes for registration and measurement. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(7), 1996.
[10.29] S. Pizer, C. Burbeck, J. Coggins, D. Fritsch, and B. Morse. Object shape before boundary shape: Scale space medial axis. *J. Math. Imaging and Vision*, 4, 1994.
[10.30] I. Rothe, H. Süsse, and K. Voss. The method of normalization to determine invariants. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(4), 1996.
[10.31] G. Sandini and V. Tagliasco. An anthropomorphic retina-like structure for scene analysis. *Computer Graphics and Image Processing*, 14, 1980.
[10.32] E. Schwartz. Computational anatomy and functional architecture of the striate cortex: a spatial mapping approach to perceptual coding. *Vision Res.*, 20, 1980.
[10.33] M. Shamos. Geometric complexity. In *7th Annual ACM Symposium on Theory of Computation*, 1975.
[10.34] D. Sinclair and A. Blake. Isoperimetric normalization of planar curves. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(8), August 1994.
[10.35] W. Snyder and I. Tang. Finding the extrema of a region. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1980.

[10.36] C. Therrien. *Decision, Estimation, and Classification*. Wiley, 1989.

[10.37] F. Tong and Z. Li. Reciprocal-wedge transform for space-variant sensing. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(5), May 1995.

[10.38] A. Veeraraghavan, A. Roy-Chowhury, and R. Chellappa. Matching shape sequences in video with applications in human movement analysis. *IEEE Trans. on Pattern Analy. and Machine Intel.*, 27(12), 2005.

[10.39] C. Weiman and G. Chaikin. Logarithmic spiral grids for image processing and display. *Computer Graphics and Image Processing*, 11, 1979.

[10.40] D. Weinshall and C. Tomasi. Linear and incremental acquisition of invariant shape models from image sequences. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(5), May 1995.

[10.41] M. Werman and D. Weinshall. Similarity and affine invariant distances between 2d point sets. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(8), August 1995.

[10.42] R. Yip, P. Tam, and D. Leung. Application of elliptic Fourier descriptors to symmetry detection under parallel projection. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(3), March 1994.

[10.43] H. Zabrodsky, S. Peleg, and D. Avnir. Symmetry as a continuous feature. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(12), December 1995.

[10.44] D. Zhang and G. Lu. A comparative study of curvature scale space and fourier descriptors for shape-based image retrieval. *Journal of Visual Communication and Image Representation*, 14(1), March 2002.

11 Representing and Matching Scenes

One of these things is not like the other.
– Sesame Street

11.1 Introduction

In this chapter rather than matching regions that we did in Chapter 10, we consider issues associated with matching scenes.

Matching at this level establishes an interpretation. That is, it puts two representations into correspondence:

- (Section 11.2) In this section, both representations may be of the same form. For example, correlation matches an observed image with a template, an approach called *template matching*. Eigenimages are also a representation for images that use the concepts of principal components to match images.
- (Section 11.3) When matching scenes, we don't really want to match every single pixel, but only do matching at points that are "interesting." This requires a definition for interest points.
- (Sections 11.4, 11.5, and 11.6) Once the interest points are identified, these sections develop three methods, SIFT, SKS, and HoG, for describing the neighborhood of the interest points using *descriptors* and then matching those descriptors.
- (Section 11.7) If the scene is represented abstractly, by nodes in graphs, methods are provided for matching graphs.
- (Sections 11.8 and 11.9) In these sections, two other matching methods, including deformable templates, are described.

As we investigate matching scenes, or components of scenes, a new word is introduced, *descriptor*. This word denotes a representation for a local neighborhood in a scene, a neighborhood of perhaps 200 pixels, larger than the kernels we have thought about, but smaller than templates. The terms kernels, templates, and descriptors, while they do connote size to some extent, are really describing how this local representation is used, as the reader will see.

11.2 Matching Iconic Representations

11.2.1 Matching Templates to Scenes

Recall that an iconic representation of an image is an image, e.g., a smaller image, an image that is not blurred, etc. In this section, we need to match two images.

A template is a representation for an image (or sub-image) that is itself an image, but almost always smaller than the original. A template is typically moved around the target image until a location is found that maximizes some match function. The most obvious such function is the sum squared error, sometimes referred to as the sum-squared difference (SSD),

$$SSD(x,y) = \left(\sum_{u=1}^{N} \sum_{v=1}^{N} (f(x+u, y+v) - T(u,v) \right)^2, \tag{11.1}$$

which provides a measure of how well the template (T) matches the image (f) at point x, y, assuming the template is $N \times N$. If we expand the square and carry the summation through, we find[1]

$$SSD(x,y) = \sum_{u,v} f^2(x+u, y+v) - 2 \sum_{u,v} f(x+u, y+v) T(u,v) + \sum_{u,v} T^2(u,v). \tag{11.2}$$

In this equation, the template indices are denoted as ranging from 1 to N. However, the origin of the template could be arbitrarily defined to be in the center of the template, and in that case, the template indices could have negative values without any loss of generality.

Let's look at these terms: The first term is the sum of the squared image brightness values at the point of application. It says nothing about how well the image matches the template (although it IS dependent on the image). The third term is simply the sum of the squared elements of the template, and is a constant, no matter where the template is applied. The second term, containing both the image and the template, is the key to matching, and that term is the correlation.

We have seen correlation many times in this book. For example, an edge detector actually looks like an edge, a fact that a kernel operator detects using correlation. Correlation occurs over and over again. Due to the image dependency in the first term of Eq. 11.2, the use of maximum of correlation for matching isn't quite the same as minimizing the SSD, and that difference can cause problems and require normalization.

Normalized Correlation

To make correlations less sensitive to variations in lighting they may be normalized by modifying both the image and the template by:

- Computing the mean of the image in the region under the template, denoted $\hat{f}(x,y)$.
- Computing the mean of all the points in the template, denoted $\hat{T}$.

[1] The relationship between correlation and squared error.

- Computing the variances

$$S_1 = \sum_{x,y} [f(x,y) - \hat{f}]^2 \tag{11.3}$$

$$S_2 = \sum_{u,v} [T(u,v) - \hat{T}]^2 \tag{11.4}$$

  where the sums are over the area of the templates. Note that $\hat{f}$ is parameterized by (x, y) because it changes as the location of the template is changed.
- Then the normalized correlation becomes

$$\frac{\sum_{u,v} [f(x+u, y+v) - \hat{f}][T(u,v) - \hat{T}]}{S_1 S_2}.$$

11.2.2 Point Matching

If an image is considered to be just a set of points with known distances between each other, one may approach the problem by assuming a 3D model of the object, and finding the transformation from 3D to 2D that best describes the observation. These techniques are beyond the scope of this book, but the reader who needs this information can refer to [11.51] and [11.3]. However, the possible applications of matching point clouds could include comparing two shapes returned by a sparse range camera. We will discuss this in Chapter 12.

11.2.3 Eigenimages

Suppose you are given a set of p images, $\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_p$, and you are faced with the question: Which of these images is most representative of the set? You could use the mathematics from section 10.3.1 to accomplish this as follows:

1. Construct a set of vectors $\{\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_p\}$ where each $\mathbf{g}_i$ is the lexicographic representation of the corresponding image minus the average image: $\mathbf{g}_i = \mathbf{f}_i - \hat{\mathbf{f}}$.
2. Calculate the covariance matrix, K, of this set by the same methods you learned in section 10.3.1.
3. Use eigenvalue techniques to obtain the eigenvectors and eigenvalues of K.
4. Suppose that among the eigenvalues, the first k values are much larger than the others. Then the corresponding eigenvectors (eigenimages), $\mathbf{e}_i$, $i = 1, \ldots, k$, play the role of principal components. Just a few of these may suffice to reconstruct the original image without losing too much information.

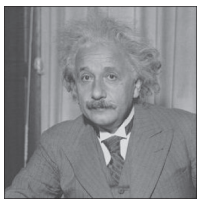
These ideas can be extended to image matching as follows:

5. Project each image onto the principal eigenvectors by constructing a matrix with those eigenvectors as columns, $[\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_k]$.

$$\mathbf{w}_i^{\mathrm{T}} = \mathbf{g}_i^{\mathrm{T}} [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_k]. \tag{11.5}$$

   Then, $\mathbf{e}_j$ is the j-th eigenimage, and $\mathbf{w}_i^{\mathrm{T}}$ is a k-dimensional vector, containing the projection coefficient of the original image onto each eigenimage.

Table 11.1 For the original images shown across the top, the two principal eigenimages are in the second row. $\mathbf{w}_i$ is the vector of projections of the i^{th} image onto the two principal eigenimages. The projection from the test image is also shown. $\mathbf{w}_{test}$ most closely resembles $\mathbf{w}_1$. Library names: facegray.png, monalisa.jpg, clock.jpg, einstein.jpg

Original images	 $\mathbf{g}_1$ $\mathbf{w}_1 = \begin{bmatrix} -5.863 \\ -10.869 \end{bmatrix}$	 $\mathbf{g}_2$ $\mathbf{w}_2 = \begin{bmatrix} 18.185 \\ -8.937 \end{bmatrix}$	 $\mathbf{g}_3$ $\mathbf{w}_3 = \begin{bmatrix} -12.321 \\ 19.806 \end{bmatrix}$
Principal components	 $\mathbf{e}_1$	 $\mathbf{e}_2$	
Test Image	 $\mathbf{w}_{test} = \begin{bmatrix} 20.23 \\ 44.28 \end{bmatrix}$		
Distance	$d(\text{Face, Test}) = 20.2$	$d(\text{Einstein, Test}) = 44.3$	

6. Given an unknown image, say $\mathbf{g}_{test}$, determine the projection of $\mathbf{g}_{test}$ onto each of the eigenvectors, producing a vector of projections similar to those resulting from Eq. 11.5.

$$\mathbf{w}_{test}^{\mathrm{T}} = \mathbf{g}_{test}^{\mathrm{T}} [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_k] \tag{11.6}$$

7. Compare the vector $\mathbf{w}_{test}$ with each of the other projection vectors, $\mathbf{w}_i$, (by computing Euclidean distance or other similar measure), and assign the unknown to the closest class.

Below, we show an interesting example of applying the eigenimage approach to face recognition [11.48]. We have 3 images in the database (`Face`, `Einstein`, and `Clock`), the image to be compared is `Monalisa`. Each image is of size 256×256. Following the seven steps described above, we first calculate the eigenimages. Table 11.1 displays the three original images and two eigenimages derived using the first two dominant eigenvectors. Because $\sum_{i=1}^{p} \lambda_i$ is close to $\lambda_1 + \lambda_2$, we conclude that two eigenimages will be sufficient. Table 11.1

also shows the projection coefficients calculated by Eq. 11.5. Based on a simple Euclidean distance calculation, it turns out that the closest match to `Monalisa` is `Face`. Intuitively, we recognize that the pose of `MonaLisa` is much closer to that of `Face` than `Einstein` in this set of images, so the decision makes sense.

Reducing Computational Complexity

Even though the eigenimage approach has potential for image matching, from the procedure described above, we see that the most time-consuming step is the derivation of the eigensystem. When the size of images is large, the calculation of the covariance matrix (which is $mn \times mn$) can take up a lot of computation resources or be completely infeasible.

One approach to reducing computation is illustrated below through an example. Assume that each image has only four pixels and that there are only three such images in the set. That is, $mn = 4$ and $p = 3$. Let the images be

$$\mathbf{f}_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \quad \mathbf{f}_2 = \begin{bmatrix} 4 \\ 1 \\ 3 \\ 2 \end{bmatrix} \quad \mathbf{f}_3 = \begin{bmatrix} 4 \\ 3 \\ 0 \\ 3 \end{bmatrix}.$$

Then the mean $\hat{\mathbf{f}} = [3, 2, 2, 3]^{\mathrm{T}}$. Subtracting the mean gives

$$\mathbf{g}_1 = \begin{bmatrix} -2 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{g}_2 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \quad \mathbf{g}_3 = \begin{bmatrix} 1 \\ 1 \\ -2 \\ 0 \end{bmatrix}.$$

Construct the matrix G, in which the i^{th} column of G is one of the images $\mathbf{g}_i$. and consider the product $S = GG^{\mathrm{T}}$. In this example,

$$G = \begin{bmatrix} -2 & 1 & 1 \\ 0 & -1 & 1 \\ 1 & 1 & -2 \\ 1 & -1 & 0 \end{bmatrix}$$

and

$$S = \begin{bmatrix} 6 & 0 & -3 & -3 \\ 0 & 2 & -3 & 1 \\ -3 & -3 & 6 & 0 \\ -3 & 1 & 0 & 2 \end{bmatrix}.$$

Observe that S is the scatter matrix, identical to the covariance except for a multiplicative scale factor. In general, S is huge – if the image is 256×256, then S is $256^2 \times 256^2$.

However, if there are only, say, three images in the set, then G is $256^2 \times 3$. Observe that the inner product, $S_2 = G^{\mathrm{T}}G$ is only 3×3. In this example,

$$S_2 = G^{\mathrm{T}}G = \begin{bmatrix} 6 & -2 & -4 \\ -2 & 4 & -2 \\ -4 & -2 & 6 \end{bmatrix}$$

S_2 turns out to have eigenvalues of $0.0, 6.0$, and 10, and a principal eigenvector of $[-0.707 \;\; 0.0 \;\; 0.707]^{\mathrm{T}}$.

Suppose μ is one of the eigenvectors of $G^{\mathrm{T}}G$. That is, μ is a vector that satisfies

$$G^{\mathrm{T}}G\mu = \lambda\mu. \tag{11.7}$$

Here's the trick: multiply on both sides of Eq. 11.7 by G to obtain

$$GG^{\mathrm{T}}(G\mu) = \lambda(G\mu), \tag{11.8}$$

for some constant λ, and we notice that $G\mu_i$ is an eigenvector of GG^{T}. Since the size of GG^{T} is dramatically larger than the size of $G^{\mathrm{T}}G$, we obtain a similarly dramatic decrease in complexity in the process of determining the eigenvectors.

Continuing the example, multiplying G by the principal eigenvector produces

$$G\mu = \begin{bmatrix} -2 & 1 & 1 \\ 0 & -1 & 1 \\ 1 & 1 & -2 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} -0.707 \\ 0.0 \\ 0.707 \end{bmatrix} = \begin{bmatrix} 2.12 \\ 0.707 \\ -2.121 \\ -0.707 \end{bmatrix}.$$

Normalization of this vector to make it a unit vector gives $[.6708 \;\; .2236 \;\; -.6708 \;\; -.2236]^{\mathrm{T}}$.

Now, if you haven't already done so, crank up MATLAB and see what the eigenvectors of GG^{T} are.

Thus, if e_i's are the eigenvectors of S, we can obtain them using simple linear algebra. Of course, we can't find *all* of them. Since G is $mn \times p$, and $p \ll mn$, the rank of GG^{T} will be at most p, and so only p of the eigenvalues will be nonzero.

So you have a tool for evaluating sets of images, and matching images to their eigenimages. Now, we move on to defining and matching "interesting" regions of images.

11.3 Interest Operators

An *interest operator* is an operator that returns a high value at those points in an image that are "interesting." There are many such operators, dating back to Moravec's [11.34] work in 1980. Here, we will study just two, the Harris-Laplace operator, and the SIFT interest operator.

11.3.1 The Harris-Laplace Operator

What is an interesting point in an image? It is certainly not a point at which there is no change in any direction! How about a point on a line or an edge? At such a point, locally, there is a specific direction in which brightness changes maximally and another direction in which the change in minimal – somewhat more interesting. But what about a corner? As shown in Figure 11.1(c), there is a strong change in brightness in multiple directions.

So a corner would be an interesting point. But how do we turn these observations into a useful algorithm? Moravec, mentioned above, did it by taking a small square window e.g. 3×3 and extracting those 9 pixels. He then compared them with the 3×3 window

(a) Regions that are homogenous or just noisy are boring.

(b) Regions with edges are a little interesting.

(c) Regions with corners are quite interesting.

Figure 11.1 The Harris interest operator defines corners as interesting. Observe that the vertical edge and the corner only look like a vertical edge and a corner from far away. Up close, one notices additional detail. This is a scale property that will be discussed next.

extracted by moving the window up, down, left, and right. At interesting points, the difference was maximal. This approach works surprisingly well, but it can be improved upon. To accomplish that, express Moravec's intuition in an equation as follows:

First, replace the words "extract a small square window" with a window function $w(x, y)$ that is zero outside the small neighborhood around the point being considered. Such a function could be a simple square of all 1's or a center-weighted function like a Gaussian. Then, characterize the change of intensity resulting from a shift of Δx, Δy:

$$E(\Delta x, \Delta y) = \sum_{x,y} w(x, y)[f(x + \Delta x, y + \Delta y) - f(x, y)]^2 \tag{11.9}$$

Something like $f(x + \Delta x) - f(x)$, is immediately recognized as potential Taylor series.

$$E(\Delta x, \Delta y) = \sum_{x,y} w(x, y) \left\{ \left[f(x, y) + \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta y + \text{Higher Order Terms} \right] - f(x, y) \right\}^2 \tag{11.10}$$

Expanding the square, moving the summations to the individual terms, and dropping the higher order terms, we can write this objective function in matrix form as

$$E = [\Delta x \ \Delta y] M \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}. \tag{11.11}$$

where, using subscript notation for partial derivatives,

$$M = \sum w(x, y) \begin{bmatrix} f_x^2 & f_x f_y \\ f_y f_x & f_y^2 \end{bmatrix} \tag{11.12}$$

At this point, we observe that M has exactly the form of a covariance matrix, prompting us to think of what that might mean. The usual covariance matrix measures the distribution of a cluster of points, x and y. Here, however, we are describing the distribution of *changes*, Δx and Δy. Remember that the magnitude of the eigenvalues of the covariance tells us how

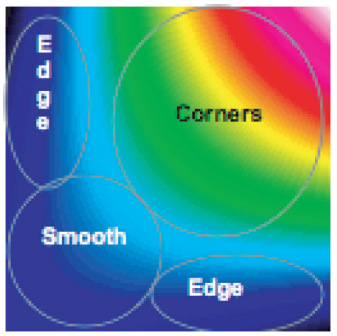


Figure 11.2 The Harris interest operator, R. The horizontal and vertical axes are the eigenvalues of M. The origin is in the lower left corner.

the data is distributed in the directions of the eigenvectors. The same thing is true here, but it is the changes in data that are being characterized. So:

- Both eigenvalues small: smooth region . . . uninteresting
- One eigenvalue much larger than the other: an edge
- Both eigenvalues large and roughly the same size: a corner . . . interesting.

It isn't particularly hard to find the eigenvalues of a 2×2 matrix; it involves solving a quadratic, which in turn involves a square root. Not hard at all, unless you do it a few million times.

Now here is an interesting observation: Consider the function

$$R = \det(M) - \alpha(\mathrm{Tr}(M))^2 \tag{11.13}$$

where α is a small constant, like 0.05.

If we graph R as a function of the two eigenvalues, we get large values when both eigenvalues are similar and large, moderate values when one is large and the other small, and very small values when both are very small – exactly what we want in a descriptor. If you recall the definitions of the determinant and the trace, you will realize that there are no square roots to take and no tests to make; large R means corner. The function R is illustrated in Figure 11.2. We refer to determining R as the *Harris operator*.

So we have (apparently) a way to find corners, but a significant detail is omitted: scale.

The Problem with Scale

Figure 11.3 shows the problem: a line is shown containing a curve. At least, it contains a curve at some scale. If we construct a Harris operator the size of the large circle, it sees a distinct corner. If, however, our operator runs over just the small circle it sees an almost perfect straight line, so we must find a way to take scale into account. The following algorithm [11.33] works well:

Looking at Eq. 11.12, we observe that there are actually two different scales in this problem. The first is the scale of the derivative itself. If we use a derivative-of-a-Gaussian approach, the σ of the Gaussian affects the scale of the features that will give a strong response to the edge detector. Second, the function w, the window function that limits the

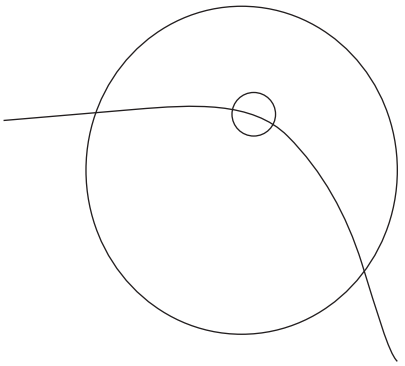


Figure 11.3 Whether a feature is a corner or a straight line depends on the scale of measurement.

area over which R is defined, also has an associated scale. This observation leads to an algorithm:

1. Construct a scale space, as described in section 5.6. Each layer of the scale space is created by blurring using a different scale. Experience [11.31] has shown that the following values work well: $\sigma_0 = 1.6$ and $k = \sqrt{2}$, where σ_0 is the initial value of scale, and k is the multiplier that determines the scale of subsequent layers of the space, as $\sigma_i = k\sigma_{i-1}$.
2. Apply the Harris operator at each point in that space. One might think that a point that is a local maximum of the Harris in both scale and space would be an interest point, but the Harris is not very precise in localizing the scale. Instead, if any point in the scale space, $R(x, y, \sigma)$ is a local maximum, its spatial coordinates x, y will be remembered. That is, (x, y) will be remembered if R is sufficiently large and satisfies $R(x, y, \sigma) > R(x', y', \sigma)$ for all (x', y') neighbors of (x, y).
3. At each point (x, y) that has been remembered as a local maximum of the Harris, search over scale for a maximum of the derivative (i.e., the Laplacian).
4. Any point found in this way that has a distinct maximum of the Laplacian and the Harris is an interest point.

The Laplacian

It is important to understand a few important characteristics of the Laplacian operator before implementing it. The Laplacian is the sum of the second derivatives, but, as has been mentioned often, we can't actually take a derivative, we can only *estimate* a derivative. Normally, we convolve with the sampled second partial derivatives of a Gaussian, and the scale of the Gaussian is important. The process of using the Laplacian of a Gaussian to create a kernel is referred to as the *Laplacian of a Gaussian* operator or LoG. Recall from section 5.4.6 that the scale (standard deviation) of the Gaussian used to formulate the kernel, the size of that kernel in pixels, and the scale of the features being sought are all related.

Thus, the Laplacian should be computed using the scale for the particular level of scale space being considered. If the scale of the feature is unknown, then the Laplacian may be applied to each level of a scale space. As one does this at each point, an extreme of the LoG will indicate the scale that is best for that point in that image.

Figure 11.4 The same scene viewed at two different scales. To apply any operator to these images, the appropriate scale must be chosen.

11.3.2 The SIFT Interest Operator

The *Scale Invariant Feature Transform* (SIFT) is one of the best-known and most highly cited algorithms in the Computer Vision literature. Two versions of it exist, [11.31, 11.32]; the second is described here:

The interest operator used in SIFT bears many similarities to the Harris-Laplace, and it also operates over a Gaussian scale space. There are, however, some important differences, but before we can explore those, we need to take a closer look at scale space.

When the derivative of a Gaussian convolution is used to find edges in an image, the user must recognize that the form of the Gaussian involves a variable, σ, which affects the performance of the algorithm. For example, consider Figure 11.4 that illustrates the same image viewed at two different scales. In order to find edges of objects of different scales, the scale, σ, of the edge finder must be appropriate.

A scale space could be used. The scale space used in SIFT [11.31], is a compromise between a pyramid and a block scale space (such as the one illustrated in Figure 5.13). This new configuration uses the concept of an *octave*, a set of images with different scales, but the same size, over which the scale doubles. Once the scale is doubled, the original image is subsampled, 2:1 in each direction, an operation equivalent to doubling the scale, and another octave is computed at this reduced size. This is illustrated in Figure 11.5. The advantage of this approach is computational efficiency. That is, after subsampling, processing the next octave takes only 1/4 the time. In the example of Figure 11.5, the octave contains images blurred with scales of 0.707, 1.0, 1.414. 2.0, and 2.83. Thus we have one octave, with scales varying from 1 to 2, plus an image with lower scale, and one with higher scale. These extra images are needed because they will be used in computing the Difference of Gaussians scale space.

Figure 11.5 Three octaves of a scale space. In order to illustrate the lower (leftmost column) scale, two of the five images have been omitted.

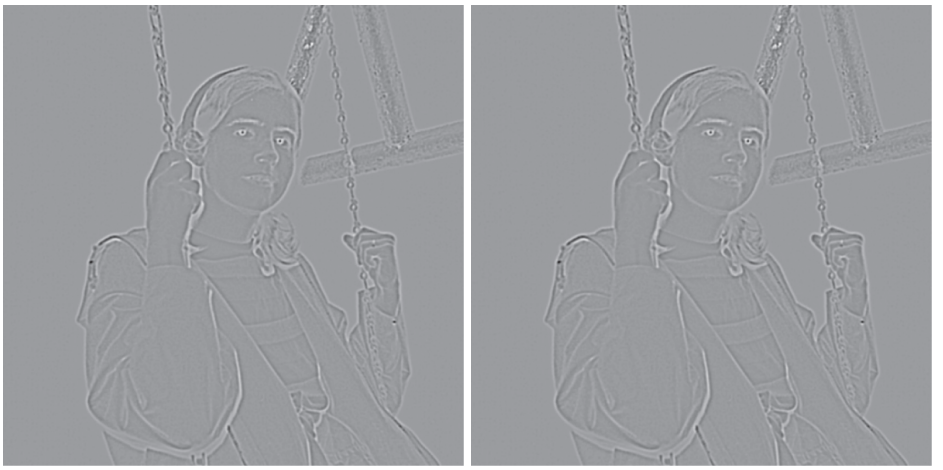


Figure 11.6 The image on the left is the Laplacian computed by adding two second derivatives, both using a scale of 1.2. The image on the right is computed by differencing two Gaussian blurred images, one having a scale of 1.0 and the second having a scale of 1.414.

SIFT uses the LoG to find interest points, but estimates the LoG by the Difference of Gaussian scale space as shown in Figure 11.7. For every level of the scale space, the next level is subtracted, pixel by pixel, producing a "DoG" (Difference of Gaussians).[2] That is, if the Gaussian scale space is denoted $f(x, y, \sigma)$, then the Difference of Gaussians, D is defined[3] by

$$D(x, y, \sigma) = f(x, y, k\sigma) - f(x, y, \sigma), \tag{11.14}$$

In every case in this chapter, the ratio of scales is $\sqrt{2}$. In Figure 11.6, a Laplacian computed with a scale of 1.2 is compared with the difference between two Gaussian blurred images, one having scale of 1.0 and one having scale of 1.414.

The DoG turns out [11.30] to be an excellent approximation to the Laplacian. The Difference of Gaussians scale space is illustrated for one octave in Figure 11.7.

The SIFT approach to interest points is to find points where the DoG is maximal in both space and scale. That is, the triple $I = [x, y, \sigma]^{\mathrm{T}}$ is an interest point if $D(x, y, \sigma)$ is a local extremum.

The SIFT interest operator is particularly efficient in applications where the DoG space must be computed anyway.

At this point in the SIFT process, we know the position of the interest points and the scale. Since we know the scale, we can calculate gradients in the neighborhood of the interest point, using the scale that we determined as the scale of the differentiating kernel. The orientation of these points is determined and a histogram of gradient directions is calculated. This histogram is a one-dimensional function of direction. It is smoothed, and the peak found. The orientation corresponding to the peak of the histogram is used to

[2] Be careful: depending on the context, the abbreviation DoG can be Difference of Gaussians, as used here, or Derivative of Gaussians, as earlier.

[3] We use the notation $f(x, y, k\sigma)$ rather than $L(x, y, k\sigma)$ as Lowe [11.31] did because we observed students confusing L with the Laplacian.

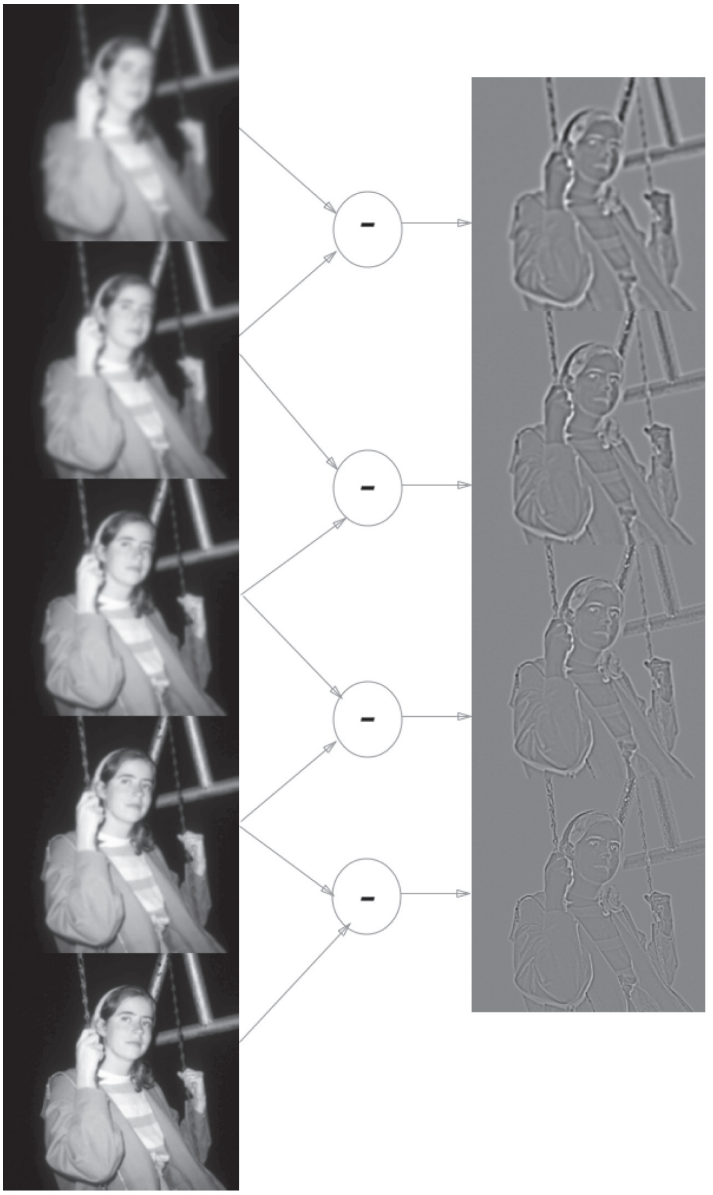


Figure 11.7 One octave of the SIFT scale space, illustrating the generation of the DoG scale space from one octave of the brightness scale space.

produce a *dominant direction* and define a coordinate system for the vicinity of the interest point.

Observe that almost everything has been done with gradients of one sort or another. Recall that making decisions based on gradients are far less sensitive to illumination variations than decisions based purely on the brightness.

At this point, the interest operator has identified the position, scale, and dominant direction of interest points in the image. In the next section, this information is used to develop a descriptor for the neighborhood of the interest point.

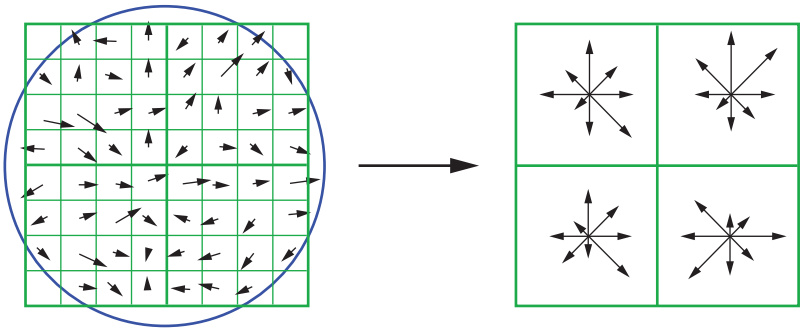


Figure 11.8 The neighborhood of the interest point is subsampled into a 16 × 16 array of points, and gradient weights and orientations are determined at each point. Then the gradients in each 4 × 4 subarray are histogrammed, and the histograms are smoothed, to construct an array of 128 numbers. Note that to simplify the figure, only a 2 × 2 array of 4 × 4 subarrays is shown here. (Figure from [11.31])

11.4 SIFT

The SIFT descriptor is a vector describing the neighborhood of the interest point. It is based on gradients, and therefore retains reduced sensitivity to illumination variations. Furthermore the descriptor is developed in scale space.

11.4.1 The SIFT Descriptor

The gradients and orientations are sampled in the neighborhood of the interest point and then weighted by a Gaussian centered at the interest point and with σ equal to the dominant scale. A 16 × 16 array of samples is extracted. The dominant direction is subtracted from each of the 256 orientations in order to create a set of direction measurements all relative to a local coordinate system defined by the dominant orientation.

For each of the 4 × 4 subregions, orientation histograms are constructed, as illustrated in Figure 11.8. Each histogram is constructed to have 8 bins (see section 8.2.2). These histograms are then smoothed. Finally, with one histogram for each of 16 subregions, a descriptor contains 4 × 4 × 8 elements.

11.4.2 Matching Neighborhoods with SIFT Descriptors

The SIFT descriptor is a single vector of length 128 that describes a small region in the vicinity of an interest point. That descriptor is constructed from gradients, and therefore is less sensitive to variations in lighting than a descriptor based on just brightness. Furthermore the descriptor is invariant to scale, translation, and rotation.

Since the descriptor is a simple vector of length 128, matching a region in one image to a region in another image can be accomplished by nothing more complicated than comparing two vectors – minimum distance classification as described in section 10.5.2.

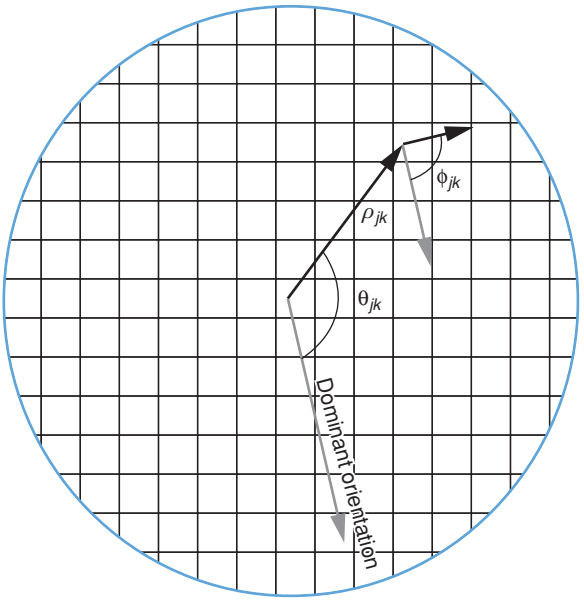


Figure 11.9 A point k in the neighborhood of interest point j is at distance ρ from the interest point. ϕ_{jk} is the gradient direction with respect to the dominant direction.

11.5 SKS

Just as SIFT finds a set of features that describe a neighborhood in a way that has many invariances, there is a version of SKS [11.24, 11.25] that performs a similar function. The principal difference is that in SKS, the matching is as important as the descriptor, whereas in SIFT, the matching process can be as simple as minimum distance between vectors.

11.5.1 The SKS Descriptor

First, interest points are detected. Any interest point detector could be used. Harris-Laplace was used in the referenced work, but others might work equally well. Determination of the interest operator also returns a characteristic scale for the neighborhood of each interest point.

A neighborhood about the interest point, say interest point j, is selected whose size is proportional to the characteristic scale. The interest point detector also returns the dominant orientation Θ_j in the circular neighborhood of radius σ_j where σ_j is the scale at which the interest point was found.

Knowledge of the dominant orientation allows construction of an "invariant coordinate system" centered at the interest point, and independent of camera orientation.

A model is now built for the neighborhood of every interest point using the direction relative to the dominant orientation, as follows:

At each point k in a circular radius σ_j around the interest point j, the ordered pair (ρ_{jk}, ϕ_{jk}) is extracted. As in Eq. 10.12, ρ_{jk} is the distance from the kth point to the interest point, as shown in Figure 11.9. In the figure, (ρ_{jk}, θ_{jk}) are the polar coordinates of point k

with respect to the invariant coordinate system at the interest point, and ϕ_{jk} is the difference between the gradient direction at point k and the dominant orientation Θ_j,

$$\phi_{jk} = \phi_k - \Theta_j.$$

Note that the features are differences in orientations, and those differences are invariant to rotation (assuming the dominant orientation is computed correctly).

Using as arguments the distance from the reference point, ρ, and the direction of local gradient, ϕ, the SKS model for a neighborhood, say neighborhood j, is

$$m_j(\rho, \phi) = \sum_{k=1}^{K} \left[\exp\left(-\frac{(\rho - \rho_{jk})^2}{2\sigma_\rho^2} \right) \exp\left(-\frac{(\phi - \phi_{jk})^2}{2\sigma_\phi^2} \right) \right], \tag{11.15}$$

The sum is taken over all the points in the neighborhood. Observe that this is almost identical to the form used in the representation of a boundary of a regions, in Eq. 10.12.

The description of the SKS algorithm here presents only two measurements, ρ and ϕ_{jk}. However, one could easily envision other measurements that might be made to characterize the points in the neighborhood (e.g. color). Adding additional measurements is easily accommodated by allowing the scalar ϕ_{jk} to be replaced by a vector ν_{jk}. The model at the feature point j is therefore more generally given by:

$$m_j(\rho, \nu) = \sum_{k=1}^{K} \left[\exp\left(-\frac{(\rho - \rho_{jk})^2}{2\sigma_\rho^2} \right) \exp\left(-\frac{||\nu - \nu_{jk}||^2}{2\sigma_\nu^2} \right) \right] \tag{11.16}$$

The summation in Eq. 11.16, may be replaced in some instances by a *max* operation. Although this changes the intuition of the model, it also works as well as and sometimes better than the summation.

The model $m_j(\rho, \nu)$ for neighborhood j can be viewed as a function that estimates the closeness of a given feature vector, ν, to all the feature vectors around the interest point j. The model function can also be precomputed and stored as a lookup table that considerably speeds up the matching process.

Equation 11.16 is still simplified a bit, because the elements of the vector ν will almost always have different variances. For this reason, in general, a covariance representation is required such as:

$$m_j(\rho, \nu) = \sum_{k=1}^{K} \left[\exp\left(-\frac{(\rho - \rho_{jk})^2}{2\sigma_\rho^2} \right) \exp\left(-\frac{\nu^{\mathrm{T}} K^{-1} \nu}{2\sigma_\nu^2} \right) \right] \tag{11.17}$$

where K here is the covariance of the vector of measurements ν.

11.5.2 Matching Neighborhoods with the SKS Descriptor

To match a neighborhood to an SKS model, we use Eq. 10.77 (repeated here for convenience) in almost the same way it was used in Chapter 10.

$$A_C(\mathbf{x}) = \frac{1}{Z} \sum_k m(||\mathbf{x} - \mathbf{x}_k||, \mathbf{v}_k) \tag{11.18}$$

Figure 11.10 The image on the left (Library name: 003a.png) and the image in the middle (Library name: 003d.png) are taken of the same area of ground. The figure on the right is produced by registering the first two and then subtracting them. The registration is essentially perfect. Had the illumination been the same, the dark area would be completely black, but because of variation in cloud cover, they were not identical, so the subtraction did not produce zero.

Since we are now matching neighborhoods rather than boundaries, the sum is over all the pixels of interest in the neighborhood instead of pixels along a line. Here, the usual feature of interest is the difference in angles, Φ_{jk} rather than the curvature that we used when considering boundaries. But other than these differences, the matching equation is interpreted identically.

Use of the SKS philosophy to match regions in aerial images is demonstrated in Figure 11.10, which illustrates two images of the same region taken with different orientations, heights, and cloud cover. The fact that everything is done with gradients rather than pixels provides almost perfect invariance to brightness variations such as cloud cover. The determination and use of the local invariant coordinate system provides invariance to rotation and translation. Experiments have shown SKS to be slightly better at dealing with such problems than SIFT. Figure 11.11 illustrates the applicability of the same matching strategy in finding correspondences by showing the correspondence number in red. Since it is difficult to see the correspondence numbers in Figure 11.11, we have enlarged the area around the stack on both images in Figure 11.12.

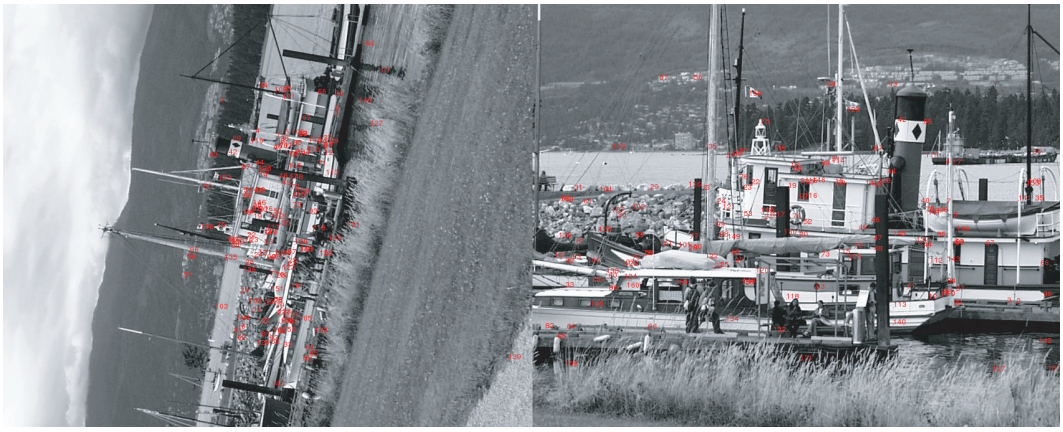


Figure 11.11 Two images of the same boat taken from different locations, with different zoom and different camera angle. The red numbers indicate correspondences as determined by the SKS algorithm described in this section.

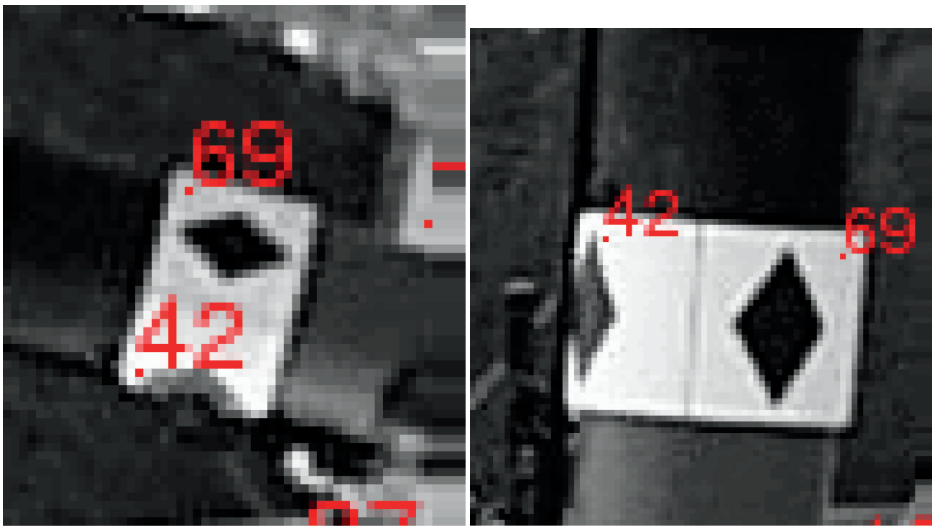


Figure 11.12 Enlargement of the same area from the two images in Figure 11.11; illustrating two corresponding points, points 42 and 69.

11.6 HoG

The *Histogram of Oriented Gradients* (HoG) descriptor [11.8] extends the use of histograms of gradient directions in SIFT to a more dense distribution of these gradient histograms. This approach has been demonstrated to be particularly good at identifying pedestrians in traffic scenes.

11.6.1 The HoG Descriptor

Dalal and Triggs, in their description of the concept of HoG, [11.8] first divide the image into "cells" of 8×8 pixels. At every pixel in the cell, the gradient magnitude and direction are computed. The cells are grouped into blocks of two cells, or sixteen pixels on a side. Each cell thus defined contains 64 pixels and each block 256. A histogram of the gradient direction is created for each cell, and each such histogram is partitioned into 9 bins over 180 degrees.

The histograms in each block are normalized to correct for contrast variation due to illumination differences (See [11.8] for details on the normalization). Overlapping blocks are used to build the final descriptor, a vector of histograms.

The HoG descriptor may be visualized as illustrated in Figure 11.13 that shows the result of HoG applied to a person sitting.

11.6.2 Matching HoG Descriptors

Since the HoG descriptors are rather long vectors, they are usually matched using *Support Vector Machines* (SVMs), one of the advanced methods in pattern recognition systems. They have excellent performance, if the problem satisfies their architecture. They are extremely well suited to answering the question "Is the thing I am looking at an X or

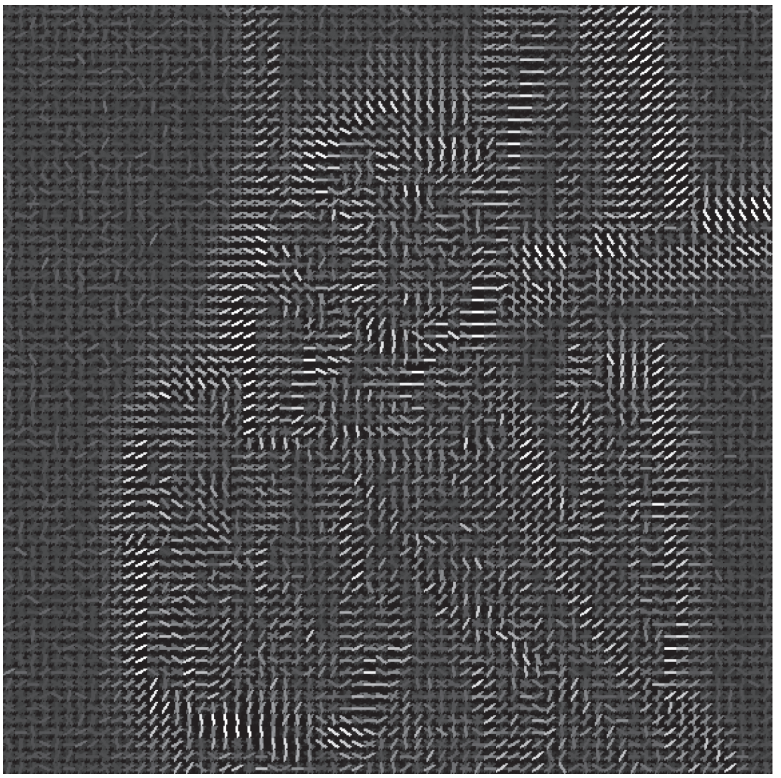


Figure 11.13 An HoG descriptor of a seated person. The histogram of each cell is represented by nine lines passing through the center of each cell, one line per histogram bin. The brightness of each line is proportional to the height of the bin, and the angle of each line is the direction repesented by each cell. Note that the lines are in the direction of the gradient, which is in the direction of the most rapid change in brightness. Often, only one line is shown, indicating that the cell is dominated by a single direction.

a Y?" However, SVMs are not really designed to address "Is the thing I am looking at an A?" In this latter case, since the SVM architecture can really only distinguish between two classes, we must change the second question to "Is the thing I am looking at an X or a SOMETHINGELSE?" This requires finding a collection of examples of SOMETHINGELSE, a challenging task.

SVMs also do not readily lend themselves to determining similarity, which is required for matching a point to one of more than two classes.

Since SVMs are pattern classification algorithms and not (per our definition) Computer Vision algorithms, we present only a brief description of them in Appendix A.

11.7 Graph Matching

In this section, we consider the problem of matching image representations that are fundamentally graph-based. However, we allow the data stored at a node in the graph to include images or templates.

Subgraph isomorphism is the most straightforward matching method. First, we define *isomorphic*. Let $G_1 =< V_1, E_1 >$ be a graph that consists of a set of vertices, V_1 and a set of edges E_1. Also, let $G_2 =< V_2, E_2 >$, be a graph that consists of a set of vertices, V_2 and a set of edges E_2. Then G_1 and G_2 are isomorphic if there exists a function $f(v)$ that maps nodes in G_1 to nodes in G_2. That is, if $v' = f(v)$, for $v \in G_1$ then $v' \in G_2$, and f will be *injective* (one-to-one). Further, if $edge(v_1, v_2) \in E_1$, then $edge(f(v_1), f(v_2)) \in E_2$.

The subgraph isomorphism problem is this: Given a pair of non-isomorphic graphs, G_1 and G_2, is there a subgraph $S =< s, e >$ of G_1 that is isomorphic to G_2?

Unfortunately, the subgraph isomorphism problem is NP-complete; that is, all known algorithms to solve it are exponential in complexity. Nonetheless, for small problems, subgraph isomorphism can be a powerful tool.

Three other approaches are described here: association graphs, relaxation labeling, and spring-loaded templates. These are methods that will produce matchings on hybrid representations, that is, those that are fundamentally graph-based but that include image information.

Matching is often the same as *labeling*. That is, we seek to pose matching as a pairwise matching of the elements of a set. That is what will be illustrated here.

Suppose we have a set of *objects*. Objects may be regions in an image, cars on a street, or anything else. It is important, however, to label each object. The label for an object could be the class it is in (flanges and gaskets), the region in a model that best matches the object region in the image, or any of a variety of other, similar problems. In most labeling problems, however, we wish to simultaneously label a *set* of objects.

Let us consider the *consistency* of labeling. For example the consistency of labeling object 1 as lion with labeling object 2 as antelope. If the only information we have is the distance between objects 1 and 2, we can say that labeling is consistent if object 1 and object 2 are far apart. If however, the two objects are close together, we would speculate that labeling object 1 as a lion would not be consistent with labeling object 2 as an antelope, for the simple reason that lions and antelope are seldom found close together (except for a few instances that usually turn out poorly for the antelope).

To formalize this, assume we have a set of objects A, a set of labels Λ and a set of *consistency functions* $r : A \times \Lambda \times A \times \Lambda \rightarrow [-1, 1]$. When the value of r is close to 1.0, the labeling is defined to be consistent, and when the value is -1, inconsistent. Zero is interpreted to simply mean no information is available. Thus the value of the consistency function

$$r(x, \lambda_1, y, \lambda_2) \tag{11.19}$$

in an example application might be

$$r(x, 1, y, 2) = 0.98, \tag{11.20}$$

and can be interpreted as meaning we have high confidence that labeling region 1 as x is consistent with labeling region 2 as y.

A "labeling" of a set of objects assigns a label to each member of the set of objects.

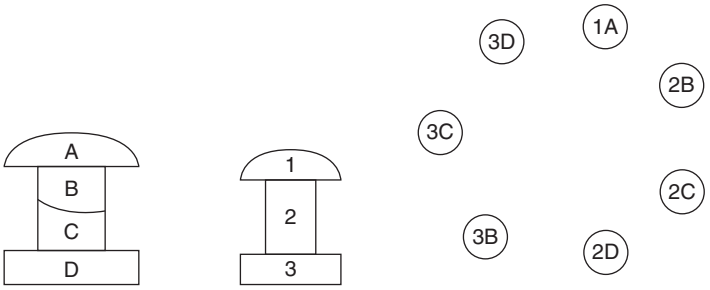


Figure 11.14 **LEFT**: A range camera has observed a scene and segmented it into segments that satisfy the same equation, however, an error has occurred. **RIGHT:** The nodes of the resulting association graph.

11.7.1 Association Graphs

Association graphs are one technique for labeling that is particularly easy to visualize.

An association graph embodies a methodology that is less restrictive than isomorphism, and that may converge more rapidly. It will converge to a solution that is consistent but not necessarily optimal (depending, of course, on the criteria for optimality used in any particular application).

The method matches a set of nodes from the model to a set of nodes extracted from the image.

Definition: An association graph is denoted $G =< V, P, R >$, where V represents a set of nodes, P represents a set of unary predicates on nodes, and R represents binary relations between nodes.

A predicate is a statement that takes on only the values TRUE or FALSE. For example let x denote a region in a range image. Then $CYLINDRICAL(x)$ is a predicate that is true or false depending on whether all the pixels in x lie on the same cylindrical surface.

A binary relation describes a property possessed by a pair of nodes, such as $ABOVE(a, b)$. Given two graphs: $G_1 =< V_1, P, R >$ and $G_2 =< V_2, P, R >$, we construct the association graph $G =< V_A, R >$ by:

- for each $v_1 \in V_1$ and $v_2 \in V_2$, if v_1 and v_2 have the same properties, construct a node of G denoted (v_1, v_2).
- If $\lambda \in R$ and $\lambda(v_1, v_1') \Rightarrow \lambda(v_2, v_2')$, add an edge to R_A that connects (v_1, v_1') to (v_2, v_2'). Here, the symbol $\Rightarrow$ means "is consistent with."

The best match of G_1 to G_2 is determined by finding the largest clique of G. Figure 11.14 RIGHT illustrates the nodes of an association graph.

Like every other technique in Computer Vision, we need to ask, "How good is this method?" Some problems arise when attempting to answer that question.

Problem 1: Is the largest clique the best match?
The largest clique is the largest set of consistent matches. Is this really the best match?

Problem 2: Computational complexity. Like the subgraph isomorphism problem, the problem of finding the largest clique is NP complete. That is, no algorithm is known that can solve this problem in less than exponential time. Is there a short cut?
Sadly, we don't have universal answers to these questions, but association graphs nonetheless are a useful tool for small consistent labeling problems.

An Example of Using Association Graphs to Match a Scene to a Model
In Figure 11.14 LEFT we illustrate an observation in which a segmentation error, oversegmentation, has occurred. That is, regions B and C are actually part of the same region, but due to some measurement or algorithmic error, have been labeled as two separate regions. In this example, the unary predicates are labels *spherical, cylindrical*, and *planar*. Regions A and 1 are spherical, while B, C, D, 2, and 3 are cylindrical. The only candidates for matches are those with the same predicate. So only A can match 1 and the only node of the association graph containing A will be (A,1).

We now construct a graph in which all candidate matches are the nodes. We then have the nodes of the association graph, as illustrated in Figure 11.14. To fill in the edges of the association graph, use is made of the concept of consistency.

Here is the challenge: Identify what it means to be consistent – determine $r(i, \lambda, j, \lambda')$, or in this example, determine $r(1, A, 2, B)$, where compatibility function r has the same meaning as described in Eq. 11.19 above. It is often easier to do this by determining what is NOT consistent, always a problem-dependent decision. Here, we define any two labelings as consistent if they do not involve the same region. Some example consistencies for this example are

$$\begin{aligned} r(1, A, 2, B) &= 1 \\ r(2, B, 2, C) &= -1 \\ r(2, B, 3, B) &= -1 \end{aligned} \tag{11.21}$$

The second line of Eq. 11.22 says that patch B in the image could not be region 2 in the model while simultaneously, patch C in the image is the same region. In both examples, inconsistencies are really based on the assumption that the segmenter is working correctly.

However, one could allow the segmenter to fail. In that case, new edges are added to the association graph, because new relationships are now consistent. For example, we could add $r(3, C, 3, D) = 1$ now, since we believe that two patches, C and D, could both be part of the same regions (the segmenter can fail by oversegmentation), however, $r(2, D, 3, D) = -1$ still holds because we still believe the segmenter will not merge patches (fail by undersegmentation). Allowing for oversegmentation produces the association graph of Figure 11.15.

Note[4] one other type of inconsistency that could be used to prevent an edge from being constructed: 3D and 3B are not connected since B and D do not border. That is, we believe that if the segmentation fails by oversegmentation, the segmenter will not introduce an entire new patch between the two. The example of Figure 11.15 does not consider this possibility. We must emphasize that how you develop these rules is totally problem-dependent!

[4] Association graphs are a kind of consistent labeling.

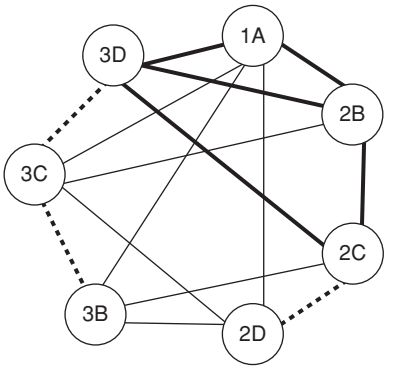


Figure 11.15 Solid lines denote the edges that are present if we assume the segmenter does not fail. The dotted lines are added if we believe the segmenter can fail by oversegmentation. The heavy lines indicate a maximal clique of size 4.

Once you have the allowable consistencies, the matching is straightforward. Simply find all maximal cliques. The maximal clique is not unique, since there may be several cliques of the same size. In this case, there are at least two maximal cliques, two of which are: {(1,A)(2,B)(2,C)(3,D)} and {(1,A)(3,B)(2,C)(2,D)}.

11.7.2 Relaxation Labeling

Using association graphs, one is constrained to integer values of the r function, corresponding to values of either "is consistent with" or "is not consistent with." In this section, we generalize this idea to provide a continuous degree of consistency. Because of the iterative algorithm used, this form of consistent labeling is known as *relaxation labeling*. Begin by defining confidence in a labeling as $P_x(\lambda)$, $P : \mathbb{R} \rightarrow [0, 1]$. Here, confidence is interpreted to mean the confidence that the label λ is correct for object x.

Now given a set of objects $X = \{x_1, x_2, \ldots, x_n\}$, and a set of labels for those objects, $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$, define a label update operation as

$$P_i^{k+1}(\lambda) = \frac{P_i^k[1 + q_i^k(\lambda)]}{\sum_j P_i^k(\lambda_j)[1 + q_i^k(\lambda_j)]} \tag{11.22}$$

where k is the iteration number. The sum is over all possible labels for object x_i. q_i is a measure of how consistent labeling x_i as λ is with all possible labelings of all objects. The denominator is a normalization to ensure P stays between zero and one.

The formula for $q(\cdot)$ is

$$q_i^{k+1}(\lambda) = \sum_j C_{ij} \left[\sum_l r(i, \lambda, j, \lambda_l) P_j^k(\lambda_l) \right]. \tag{11.23}$$

Notice in Eq. 11.23 that the consistency function r is multiplied by the strength of the labeling being considered. Therefore a labeling with low confidence does not affect the result very much, although it may be very consistent. The term C_{ij} quantifies the impact of object i on object j. That is, suppose those two objects are far apart in the image. Then, it

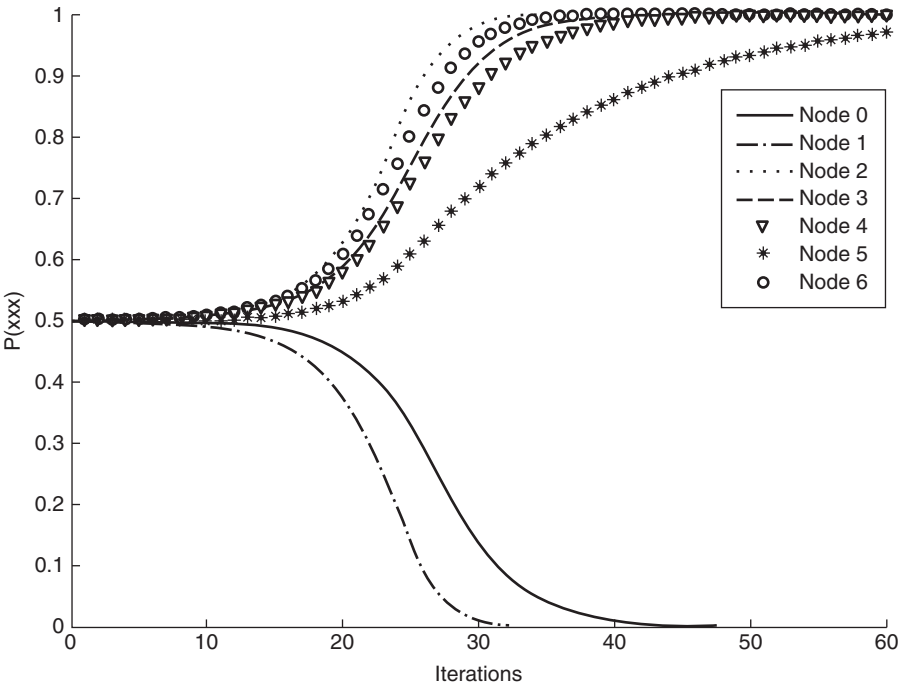


Figure 11.16 An example of consistent labeling a set of sensor nodes as "type 1" or "type 2." Over iterations, the confidence of each labeling converges to one or zero.

may be that the labeling of object i has no impact at all on the labeling of object j, and this information can be represented by simply making C_{ij} small.

Iterating over k will produce a solution that will find best labelings for the set of objects. Figure 11.16 illustrates a system with seven objects as this system interacts through repeated applications of Eqs. 11.22 and 11.23. Each labeling converges over iterations to zero or one, resulting in an optimal consistent labeling.

11.7.3 Springs and Templates

Another approach to the matching problem is provided by the *Springs and Templates* philosophy [11.14]. This is a hybrid model-matching paradigm that involves both graph- structured matching and template matching. The model is a set of rigid "templates" connected by "springs" that characterize how much deformation must be applied to the model to make it match the image. Figure 11.17 illustrates this concept using a simple model of a human face. Specific features such as eyes are matched by iconic methods such as template matching. However, in order to match an entire face, distances between the best matching locations of templates are also recorded. A match is then based on minimizing a total cost, as follows.

$$Cost = \sum_{d \in templates} TemplateCost(d, F(d)) + \sum_{d,e \in ref \times ref} SpringCost(F(d), F(e)) + \sum_{c \in missing} MissingCost(c). \quad (11.24)$$

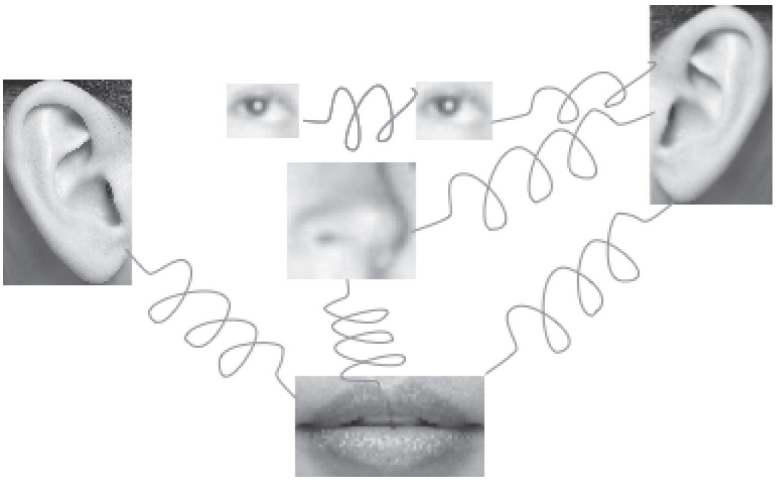


Figure 11.17 A springs-and-templates model of a face. The degree of stretching or compressing of a spring quantifies the degree of mislocation of the template. Not all springs are shown in this figure.

In Eq. 11.24, d is a template and $F(d)$ is the point in the image where that template is applied. TemplateCost is therefore a function indicating how well a particular template matches the image when applied at its best matching point. SpringCost is a measure of how much the model must be distorted (the springs stretched) to apply those particular templates at those particular locations. Finally, it may be that not every template can be located – in some images, the left eye may not be visible – and a cost may be imposed for things missing. All these costs are empirically determined. However, once they are determined, it becomes relatively easy to determine how well any given image matches any given model.

There is one significant problem with spring matching: the number of elements matched affects the magnitude of the cost. It is possible that there may be many small and truly insignificant errors. Nonetheless, these errors are summed and, because there are many, may sum to a large numerical error value. Such an image might be compared with an image that has only one error, but that error might be huge and obvious. Simply comparing errors may give intuitively incorrect results.

This is a problem that is not unique to Springs and Templates. The usual solution is to normalize the calculations, using a technique such as

$$cost = \frac{SpringCost \text{ (unary and binary)}}{Total\ number\ of\ springs} + \frac{constant}{Total\ number\ of\ references\ matched} \tag{11.25}$$

Though Springs and Templates serves as a wonderful example for illustrating both the need for normalization, and for combining methods, it has been supplanted by other, more sophisticated methods, which are mentioned in the following sections.

11.8 Springs and Templates Revisited

To solve the correspondence problem, we may use a consistency-based philosophy. In this section, we describe this philosophy through an example, which we will use again when we talk about Random Sample Consensus (RANSAC) in Chapter 12.

Figure 11.18 Boundary of an object. Points where the sign of the curvature changes are marked. Figure redrawn from [11.22].

The first step is to identify feature points that are relatively distinctive. Then the algorithm will make use of relationships between these points. Points on the boundary of a region where the curvature of the boundary changes sign meet this requirement, as illustrated in Figure 11.18.

This example derivation was first described by Shapiro and Brady [11.46] who use eigenvector methods as follows:

As in the original Springs and Templates formulation, we are finding which of a collection of feature point sets best matches one particular set. Let d_{ij} be the Euclidean distance between feature points x_i and x_j, and construct a matrix of weights

$$H = H_{ij}, \text{ where } H_{ij} = \exp\left(-\frac{d_{ij}}{2\sigma^2}\right). \tag{11.26}$$

The matrix H is diagonalized using eigendecomposition into the product of three matrices

$$H = E\Lambda E^{\mathrm{T}}. \tag{11.27}$$

Let us assume the rows and columns of E and Λ are sorted so that the eigenvalues are sorted along the diagonal in decreasing size. We think of each row of E as a feature vector, denoted $\mathbf{F}_i$. Thus $E = \begin{bmatrix} \mathbf{F}_1 \\ \cdots \\ \mathbf{F}_m \end{bmatrix}$.

Suppose we have two images, f_1 and f_2, and then suppose f_1 has m feature points while f_2 has n feature points, and suppose $m < n$. Then by treating each set of feature points independently, we have $H_1 = E_1\Lambda_1 E_1^{\mathrm{T}}$ for image f_1, and $H_2 = E_2\Lambda_2 E_2^{\mathrm{T}}$ for image f_2. Since the images have different numbers of points, the matrices H_1 and H_2 have different numbers of eigenvalues. We therefore choose to use only the most significant k features for comparison purposes.

It is important that the directions of the eigenvectors to be matched be consistent, but changing the sign does not affect the orthonormality. We choose E_1 as a reference and then orient the axes of E_2 by choosing the direction that best aligns the two sets of feature vectors. See [11.46] for details. After aligning the axes, a matrix Z characterizing the match between image 1 and image 2 is defined by

$$Z_{ij} = (\mathbf{F}_{i1} - \mathbf{F}_{j2})^{\mathrm{T}}(\mathbf{F}_{i1} - \mathbf{F}_{j2}). \tag{11.28}$$

The best matches are indicated by the elements of Z that are the smallest in their row and column.

For further literature on similar algorithms, see Sclaroff and Pentland [11.44] and Wu [11.50].

11.9 Deformable Templates

Recall deformable contours (snakes) from section 8.5. If we think of the region surrounded by a snake, rather than the snake itself, we realize we have a region whose shape can be deformed, and thus have invented "deformable templates." Objects can be tracked utilizing a philosophy that allows for deformation of the template[11.52]. In addition, the concepts of deformable templates may be useful in image database access. For example, Bimbo and Pala [11.5] compare shapes in the image with a user-drawn sketch, an "iconic index", which is a type of deformable template.

The best matching template is written as

$$\Phi(s) = \tau(s) + \theta(s), \tag{11.29}$$

where s is (normalized) arc length, τ is the template as stored in the database, and $\theta(s)$ is the deformation required to make this particular template match a sequence of boundary points in the image being accessed. We emphasize that $\theta(s)$ is the *difference* between the original and the deformed template. The image in the database that best matches the template is the image that minimizes "the difference between the original and the deformed template." This can be achieved by minimizing

$$E = \int_0^1 \left(\alpha \left[\left(\frac{d\theta_x}{ds} \right)^2 + \left(\frac{d\theta_y}{ds} \right)^2 \right] + \beta \left[\left(\frac{d^2\theta_x}{ds^2} \right)^2 + \left(\frac{d^2\theta_y}{ds^2} \right)^2 \right] - I_e(\phi(s)) \right) ds \tag{11.30}$$

where the first term represents how much the template had to be strained to fit the object, the second term represents the energy spent to bend the template, and the third term represents a measure of the gradient of the image (not the template) at the point $\phi(s)$. This is thus a deformable templates problem. The optimization problem may be solved numerically [11.5].

11.10 Conclusion

In this chapter, the student has seen a few of the representation and matching methods and seen them in sufficient depth to grasp the usefulness of each method to particular classes of problems. However, the list of methods presented here is far from exhaustive. In the following, we provide additional references related to topics discussed in this chapter.

Eigenimages are lower dimensionality representations of the original images, where the projections are chosen by minimizing the error between the original data and the projected data. For more efficient calculations in using eigenimages, readers are referred to [11.35, 11.36].

In terms of interest point detectors, our discussion only focused on 2D techniques. In order to process, e.g., video sequences, 3D interest operators have also been developed,

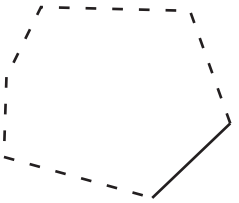


Figure 11.19 A set of points produced by an edge detector that might come from a circle or a polygon.

including the Harris3D detector [11.27], the Cuboid detector [11.12], the Hessian detector [11.49], and simply the dense sampling [11.13, 11.23]. Wang et al. [11.47] conduct a comprehensive evaluation of various detectors in the context of action recognition.

Besides HoG, SIFT, and SKS, other popularly used feature descriptors in both 2D and 3D applications include the Cuboid [11.12], the speed-up robust features (SURF) [11.2, 11.1], HoG3D [11.45], and the local binary pattern (LBP) [11.37] along with its many variants.

In section 11.7.2, we only briefly touch on a discipline called *consistent labeling*, which hypothesizes that the most consistent labeling/match is most likely to be the best. Consistent labeling has been used for many applications in vision, and in other applications such as sensor networks [11.7], where the problem is to identify the least consistent responses from a collector of sensor nodes. The original paper on association graphs was [11.40]. For further literature in consistent labeling, see [11.21, 11.39, 11.42], just to name a few.

Similarly, there are variations on deformable models [11.10, 11.11], especially for range images [11.20].

As we conclude the discussion in this chapter, the student might ask the question, "What model should I use?" Consider Figure 11.19: Should you match it to a circle or a six sided polygon? Clearly, there is no simple answer to this question. If you have prior, problem–specific knowledge that you are always dealing with circular objects, you might choose to use the circular model, which is certainly less complex than that of a polygon.

The Minimum Description Length (MDL) paradigm states that the optimal representation for a given image may be determined by minimizing the combined length of the encoding of the representation and the residual error. Interestingly, the MAP representation can be shown [11.9, 11.28] to be equivalent to the MDL representation where the prior truly represents the signal. Schweitzer [11.43] uses the MDL philosophy to develop algorithms for computing the optic flow, and Lanterman [11.26] uses it to characterize infrared scenes in ATR applications, stating "if there are several descriptions compatible with the observed data, we select the most parsimonious."

Rissanen [11.41] thinks about information theory in the MDL contex, suggesting that the quality of an object/model match could be represented by

$$L(x, \theta) = -\log_2 P(x|\theta) + L(\theta) \tag{11.31}$$

where x is the observed object, θ is the model, represented as a vector of parameters, $P(x|\theta)$ is the conditional probability of making this particular measurement given the model, and $L(\theta)$ denotes the number of bits required to represent the model. The logarithm of the

conditional probability is then a measure of how well the data fits the model. We thus may trade off a more precise fit of a more complex model with a less accurate fit of a simpler model [11.6].

Ultimately, the Computer Vision problem is not going to be solved by one program, one algorithm, or one set of mathematical concepts. Ultimately, its solution will depend on the ability to build systems that integrate a collection of specialized programs. The jury is still out on how to accomplish this and only a few authors have undertaken this formidable task. For example, Grosso and Tistarelli [11.18] combine stereopsis and motion. Bilbro and Snyder [11.4] fuse luminance and range to improve the quality of the range imagery, and Pankanti and Jain [11.38] fuse stereo, shading, and relaxation labeling. Zhu and Yuille [11.53] incorporate the MDL approach, including active contours and region growing, into a unified look at segmentation.

We omitted a discussion of Artificial Neural Networks (ANNs) from this book because the discipline of ANNs is so wide it simply cannot fit. The student is strongly encouraged to augment his/her understanding in Computer Vision by taking courses in Statistical Pattern Recognition (first) and then Artificial Neural Networks. There are numerous textbooks discussing use of neural networks for matching applications [11.19, 11.17].

The recent development of Deep Learning has largely advanced the state-of-the-art in representation and recognition. In the deep learning network, a set of features are automatically derived from the raw inputs as compared to the "engineered" features extracted using techniques discussed in Chapter 10 and this chapter. Gonzalez and Woods, in their latest edition of the Digital Image Processing book [11.15] have a thorough discussion on this matter. The most recent paper by LeCun, Bengio, and Hinton, published at the *Nature Magazine* [11.29] and the most recent book by Goodfellow, Bengio and Courville [11.16] are both good starting points for studying deep learning.

11.11 Assignments

Assignment 11.1: You are the product quality manager for a small manufacturing company. Your company makes α's and β's. Each part may be described by its length. You have taken lots of measurements and determined that the average length for α's is 10m, and for β's is 8m.

You have an annoying engineer working for you who also calculated (without being asked to) the variance in length for αs and βs, obtaining for α's a σ^2 of 0.4, and for β's a σ^2 of 2.

An unknown object comes down the assembly line. The sensor measures it and returns a value of 8.6m. Your computer immediately returns a decision that this object is an α, since 8.6 is closer to 8 than to 10. You see the result, smile, and walk away happy, knowing that your system is returning an answer that seems to make sense. The annoying engineer throws down his hat (purchased at company expense) and angrily walks away, in the other direction.

What is going on?

Assignment 11.2: You have seen Eq. 11.30 in another chapter of this book. What is the relationship between the two appearances?

Assignment 11.3: SIFT is called the SCALE INVARIANT feature transform. Are there any other things to which the SIFT descriptor is invariant, or nearly invariant? Consider variations in image brightness, translation, rotation, and possibly others.

If you claim SIFT is invariant to one of these deformations, explain why.

Assignment 11.4: Both versions of SKS appear to be invariant to translation because all the distances are relative to a fixed reference point. Are there any other things to which the SKS descriptor is invariant, or nearly invariant?

Consider variations in image illumination, translation, rotation, and possibly others.

If you claim the SKS descriptor is invariant to one of these deformations, explain why.

Assignment 11.5: In this chapter, we stated that the problem of finding the largest clique is NP-complete. What does that really mean? Suppose you have an association graph with ten nodes, interconnected with 20 edges. How many tests must you perform to find all cliques (which you must do in order to identify which of these are maximal)? You *are* permitted (encouraged!) to look up clique-finding in a graph theory text.

Assignment 11.6: In section 11.7.1, an example problem is presented that involves an association graph that allows for segmentation errors. The result of that graph is two maximal cliques, which (presumably) mean two different interpretations of the scene. Describe in words these two interpretations.

Assignment 11.7: Following is an incomplete citation:

C. Olson, "Maximum Likelihood Template Matching,"

First, locate a copy of this paper. You may use a search engine, the Web, the library, or any other resource you wish. In that paper, the author does template matching in a different way: using a binary (edge) image and a similar template, he does not ask, "Does the template match the image at this point?" Instead he asks, "At this point, how far is it to the nearest edge point?"

How does he perform this operation, apparently a search, efficiently?

Once he knows the distance to the nearest edge point, how does he make use of that information to compute a quality of match measure?

Assignment 11.8: In an image matching problem, we have two types of objects, lions and antelope (which occupy only one pixel each).

- A scene may contain only lions and/or antelope.
- Lions hunt in packs, so if you see one lion, you will see at least one other lion, usually about 5 pixels away.
- Antelope stay as close to one another as possible.
- Except for certain rare, and (for the antelope) unpleasant events, lions and antelope are VERY far apart.

We wish to use relaxation labeling to solve this assignment problem. All the formulae are in the book except the formula for the consistency $r(a, \lambda_1, b, \lambda_2)$, where a and b are points of interest in the image and the lambdas are labels for either "antelope" or "lion." Invent an r function for this problem. That is, tell how to compute values for

- $r(a, lion, b, antelope)$
- $r(a, lion, b, lion)$

- $r(a, antelope, b, lion)$
- $r(a, antelope, b, antelope)$

Assignment 11.9: Do you think the concepts of Springs and Templates would be applicable to Assignment 11.8? Discuss.

Assignment 11.10: Still thinking about lions and antelope, you observe this scene:

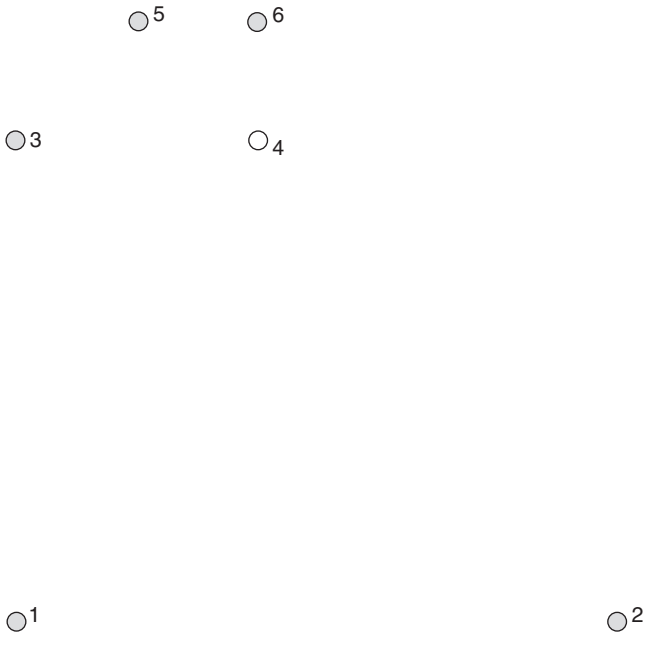


Table 11.2 Distances between pairs of animals. Also shown are interpretations (P is possible, converted to C–consistent at the end)

pair		distance	AA	AL	LA	LL
1	2	5	P	P	P	C
1	3	4	P	X	X	P
1	4	4.47	P	X	X	P
1	5	5.09	P	P	P	P
1	6	5.38				
2	3	6.4	P	P	P	P
2	4	5	P	P	P	P
2	5	6.4	P	P	P	P
2	6	5.83	P	P	P	P
3	4	2	C	X	X	X
3	5	1.4	C	X	X	X
3	6	2.2	C	X	X	X
4	5	1.4	C	X	X	X
4	6	1				
5	6	1				

For your convenience, we have, in the table above, tabulated the distance between each pair of animals. Lions are yellow (which is denoted by a gray interior) or brown (denoted by a black interior – that's right; there aren't any). Antelopes are white (denoted by a white interior) or yellow. You wish to use association graph methods to solve this problem; and since this technique is not as powerful as nonlinear relaxation, you talked to a botanist[5] who gave you some improved information: Lions *never* get closer to each other than 3 pixels, and antelope never get further from all the other antelopes than 3 pixels.

Draw an association graph for this problem. (Denote the nodes in the association graph by pairs: e.g. 1L means "interpret node 1 as a lion.") Indicate the maximal clique by circling the nodes in that clique.

Bibliography

[11.1] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. Speeded up robust features (SURF). *Computer Vision and Image Understanding*, 110(3), 2008.

[11.2] H. Bay, T. Tuytelaars, and L. V. Gool. SURF: Speeded-up robust features. In *European Conf. on Computer Vision (ECCV)*, volume 3591, pages 404–417, 2006.

[11.3] B. Bhanu and O. Faugeras. Shape matching of two dimensional objects. *IEEE Trans. Pattern Anal. and Machine Intel.*, 6(2), 1984.

[11.4] G. Bilbro and W. Snyder. Fusion of range and luminance data. In *IEEE Symposium on Intelligent Control*, August 1988.

[11.5] A. Bimbo and P. Pala. Visual image retrieval by elastic matching of user sketches. *IEEE Trans. Pattern Anal. and Machine Intel.*, 19(2), 1997.

[11.6] J. Canning. A minimum description length model for recognizing objects with variable appearances (the vapor model). *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(10), 1994.

[11.7] C. Chang, W. Snyder, and C. Wang. Secure target localization in sensor networks using relaxation labeling. *Int. J. Sensor Networks*, 1(1), 2008.

[11.8] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *International Conference on Computer Vision and Pattern Recognition (CVPR '05)*, 2005.

[11.9] T. Darrell and A. Pentland. Cooperative robust estimation using layers of support. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(5), 1995.

[11.10] D. DeCarlo and D. Metaxas. Blended deformable models. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(4), 1996.

[11.11] S. Dickinson, D. Metaxas, and A. Pentland. The role of model-based segmentation in the recovery of volumetric parts from range data. *IEEE Trans. Pattern Anal. and Machine Intel.*, 19(3), 1997.

[11.12] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie. Behavior recognition via sparse spatio-temporal features. In *IEEE Int. Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, pages 65–82, 2005.

[11.13] L. FeiFei and P. Perona. A Bayesian hierarchical model for learning natural scene categories. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[11.14] M. Fischler and R. Elschlager. The representation and matching of pictoral structures. *IEEE Transactions on Computers*, 22(1), Jan 1973.

[11.15] R. Gonzalez and R. Woods. *Digital Image Processing*. Pearson, 4th edition, 2018.

[5] Yes, a botanist! He was out of his field.

[11.16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.
[11.17] D. Graupe. *Principles of Artificial Neural Networks*. World Scientific, 2007.
[11.18] E. Grosso and M. Tistarelli. Active/dynamic stereo vision. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(9), 1995.
[11.19] S. Haykin. *Neural Networks and Learning Machines*. Prentice-Hall, 2009.
[11.20] M. Hebert, K. Ikeuchi, and H. Delingette. Spherical representation for recognition of free-form surfaces. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(7), 1995.
[11.21] R. Hummel and S. Zucker. On the foundations of relaxation labeling processes. *IEEE Trans. Pattern Anal. and Machine Intel.*, 5(5), 1983.
[11.22] K. Sohn, J. Kim, and S. Yoon. A robust boundary-based object recognition in occlusion environment by hybrid Hopfield neural networks. *Pattern Recognition*, 29(12), December 1996.
[11.23] F. Jurie and B. Triggs. Creating efficient codebooks for visual recognition. In *Int. Conf. on Computer Vision (ICCV)*, 2005.
[11.24] K. Krish, S. Heinrich, W. Snyder, H. Cakir, and S. Khorram. A new feature based image registration algorithm. In *ASPRS 2008 Annual Conference*, April 2008.
[11.25] K. Krish, S. Heinrich, W. Snyder, H. Cakir, and S. Khorram. Global registration of overlapping images using accumulative image features. *Pattern Recognition Letters*, 31(2), January 2010.
[11.26] A. Lanterman. Minimum description length understanding of infrared scenes. *Automatic Target Recognition VIII, SPIE*, 3371, April 1998.
[11.27] I. Laptev and T. Lindeberg. On space-time interest points. *International Journal of Computer Vision*, 64(2/3), 2005.
[11.28] Y. Leclerc. Constructing simple stable descriptions for image partitioning. *Inter-national Journal of Computer Vision*, 3, 1989.
[11.29] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553), 2015.
[11.30] T. Lindeberg. Image matching using generalized scale-space interest points. *Journal of Mathematical Imaging and Vision*, 52(1), 2015.
[11.31] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 20, 2004.
[11.32] D. G. Lowe. Object recognition from local scale-invariant features. In *Proc. of the International Conference on Computer Vision ICCV*, 1999.
[11.33] C. Mikolajczyk and K. Schmidt. Indexing based on scale invariant interest points. In *Eighth IEEE International Conference on Computer Vision*. IEEE, 2001.
[11.34] H. Moravec. Rover visual obstacle avoidance. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1981.
[11.35] H. Murakami and B. Kumar. Efficient calculation of primary images from a set of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(5):511–515, September 1982.
[11.36] H. Murase and M. Lindenbaum. Partial eigenvalue decomposition of large images using the spatial temporal adaptive method. *IEEE Transactions on Image Processing*, 4(5), May 1995.
[11.37] T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24, 2002.
[11.38] S. Pankanti and A. Jain. Integrating vision modules: Stereo, shading, grouping, and line labeling. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(9), 1995.
[11.39] M. Pelillo and M. Refice. Learning compatibility coefficients for relaxation labeling processes. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(9), 1994.

[11.40] R. Bolles. Robust feature matching through maximal cliques. *Proc. Soc. Photo-opt. Instrum. Engrs.*, 182, April 1979.

[11.41] J. Rissanen. A universal prior for integers and estimation by minimum description length. *Ann. Statistics*, 11, 1983.

[11.42] P. Sastry and M. Thathachar. Analysis of stochastic automata algorithm for relaxation labeling. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(5), 1994.

[11.43] H. Schweitzer. Occam algorithms for computing visual motion. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(11), 1995.

[11.44] S. Sclaroff and A. Pentland. Model matching for correspondence and recognition. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(6), 1995.

[11.45] P. Scovanner and M. Shah. A 3-dimensional SIFT descriptor and its application to action recognition. In *ACM Int. Conf. on Multimedia*, 2007.

[11.46] L. Shapiro and J. M. Brady. Feature-based correspondence: An eigenvector approach. *Image and Vision Computing*, 10(5), June 1992.

[11.47] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid. Evaluation of local spatio-temporal features for action recognition. In *British Machine Vision Conf. (BMVC)*, 2009.

[11.48] X. Wang and H. Qi. Face recognition using optimal non-orthogonal wavelet basis evaluated by information complexity. In *International Conference on Pattern Recognition*, volume 1, pages 164–167, August 2002.

[11.49] G. Willems, T. Tuytelaars, and L. V. Gool. An efficient dense and scale-invariant spatio-temporal interest point detector. In *European Conf. on Computer Vision (ECCV)*, 2008.

[11.50] Q. Wu. A correlation-relaxation-labeling framework for computing optical flow – template matching from a new perspective. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(9), 1995.

[11.51] M. Yang and J. Lee. Object identification from multiple images based on point matching under a general transformation. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(7), 1994.

[11.52] Y. Zhong, A. Jain, and M. Dubuisson-Jolly. Object tracking using deformable templates. *IEEE Trans. Pattern Anal. and Machine Intel.*, 22(5), May 2000.

[11.53] S. Zhu and A. Yuille. Region competition: Unifying snakes, region growing, and Bayes/MDL for multiband image segmentation. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(9), 1996.

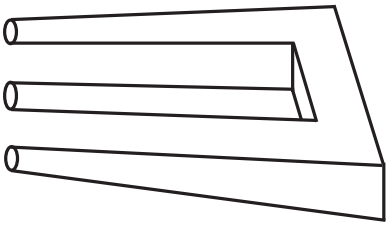
Part IV

The 2D Image in a 3D World

We humans and our computers live in a three-dimensional world, but our eyes see only projection of that world onto our two-dimensional retinas. We correct for that projection just a bit by using stereo vision. Somehow our eye-brain system knows how to infer a degree of depth by using two images together.

Of course, our eye-brain system can be fooled.

In this part of the book, we provide the student with basic tools and understanding of this phenomenon.



12 Relating to Three Dimensions

I've always been passionate about geometry and the study of three-dimensional forms.
– Erno Rubik

12.1 Introduction

Most of the images we see are projections of surfaces in the three-dimensional world around us. They result from light reflected off surfaces in that 3D world, passing through the lens of a camera, and intersecting the focal plane of the camera. These images that result from light reflected in this way are called "luminance" or "brightness" images in this book.

What was not described earlier in this book is the relationship between the 3D world and 2D images, including matching of one to another. We begin by reviewing the geometry of a simple projective camera and relating it to position of points in three dimensions. What we would really like is a range image from every scene, but that isn't always feasible, so we look at several aspects of the 2D–3D relationship:

- (Section 12.2) It is necessary to determine the 3D position of a point in space that is seen by two cameras, assuming the two cameras are known. This is the problem commonly known as *stereopsis*. When we say a camera is "known" we mean we know where it is, which way it is pointing, and all its internal parameters like focal length, resolution, and others.
- (Section 12.3) Actually it is not really necessary to know all about the cameras. Almost all the relevant information about both cameras can be determined if there are several points in each camera view that can be put into correspondence. An adequate solution to this problem leads to a wonderful little matrix called the *fundamental matrix* that contains all we need to know for the two-camera problem. Underlying this work is the correspondence problem, the problem of identifying which point in one image corresponds to which point in the other image. Finding a robust solution to the correspondence problem may be difficult.
- (Section 12.4) Once we have an approach to the correspondence problem, we can do partial matching of images and image stitching.
- (Section 12.5) If, instead of two cameras, we have one camera and a controllable light source, we can still find the 3D location of points in space. We address controllable lighting and how to achieve range imaging in this context.

- (Section 12.6) With only one camera, we can sometimes still look at how things in the image change (things like shading, texture, focus, etc.) and get 3D information. These "shape from" methods are covered in Section 12.6.
- (Section 12.7) Finally, we examine the mathematics of the three-dimensional surfaces themselves.

We should emphasize that in this book, we only introduce the student to these methods. There are ways to combine information from multiple cameras, multiple views with the same camera, and a variety of other related approaches to these problems. The student confronting a problem in the field may have need of these extensions to the basic methods of this chapter, and is referred to more detailed references such as [12.15].

Before proceeding to the next section, we clarify some definitions first:

A **range image** is a data structure in the form that you are used to, $f(x, y)$, but the difference is this: f does not represent brightness. Instead it represents some measure of distance from a camera to a surface. In Chapter 4, we distinguished between a *range image* and a *depth image*, and the difference was in the nature of the coordinate system. To be totally correct, we said, a range image is a collection of distances from a single point, the laser, to various points on the surface of interest, and hence takes the form $r(\theta, \phi)$, where θ and ϕ are the pointing angles of the laser. That image can be converted to an image of the form $z(x, y)$, which we called a depth image.

From here on, we won't distinguish between depth images and range images. Instead we will refer to both as range images. This allows us some simplicity in discussion and agreement with most literature. So from here on, when you read "range," know that it could be either measurement.

A **dense range image** is one in which, for every coordinate pair (x, y), there is a value of f given. In a dense range image the concept of neighbor makes sense (unless, of course, f is discontinuous).

A **sparse range image** is one in which f is only measured at a relatively few (x, y) points.

Finally, a **point cloud** is simply a set of triples $\{(x_1, y_1, z_1), \ldots, \}$. In a point cloud, neighborhood information is not implied, but it may be possible to compute which points are neighbors.

Frequently in literature the term "range image" means "dense range image." So if it matters (and it usually does not), be careful in interpreting this particular term.

To repeat: a range image is the distance to a surface. We are measuring distances in 3-space, but that measurement stops when we encounter the surface. We do not look inside the object. If one had a sensor that could look inside the object, (e.g., computed tomography, positron emission tomography, magnetic resonance imaging), one could represent the object in the form $d(x, y, z)$, where d denotes density or some other property of the material. Such images are common in medical imaging. However, we will not cover 3D images in this book.

12.2 Camera Geometry – Range from Two Known Cameras (Stereopsis)

In this section, we approach two issues: First we consider the camera, and relate points in the image (a 2-space) to the 3-space objects that are being imaged. Second, we find a transform that relates one image to another. Of course, we want to teach you about images and how they relate to the world, but we also want to teach you how to use several very important mathematical operations essential to Computer Vision. You will come to understand projective transformations and associated linear algebra, as well as how to minimize a quadratic function of a vector. That will involve a bit of matrix manipulation and differentiation.

First, let's consider projection.

12.2.1 Projections

We live in a *3-space*: specifically a space with three spatial dimensions – we don't consider time as a dimension (here). From this 3-space, we have the ability to "project" information from the 3-space to any of an infinite set of two-dimensional spaces we call "pictures."

Orthographic Projections

The first type of projection we could consider is called *orthographic*, and the simplest orthographic projection would be from some point in 3-space onto the $z = 0$ plane. Given a point $[x, y, z]^{\mathrm{T}}$ in space, the orthographic projection onto the $z = 0$ plane is just $[x, y]^{\mathrm{T}}$. That's pretty simple, right? Of course, when you realize that all the points with the same x and y coordinates project to the same point on the camera plane, and, relative to the world, your camera is pretty small, you realize that the orthographic projection process could end up throwing away a lot of data.

Orthographic projection is commonly used in chest x-ray, where the image obtained from the film or sensing device is almost exactly the same size as the object being imaged.

We won't do much with orthographic projections because they require extremely large focal planes. However, there is one case where we can think about such projections, and that is the case that the objects of interest are very small and a very long way from the camera (assuming a regular, projective camera model). In this case, the rays are nearly parallel, and the projection, at least the projection of the single, small object, is very nearly orthographic. In such a case, we might consider using orthographic projection because the math ends up being a bit easier.

However, let's go ahead and think about more realistic cameras. To understand cameras, recognize that there are two coordinate systems of interest, the *world coordinate system* that represents the camera in the world, and the *camera coordinate system* that represents the location of pixels on the focal plane of the camera. We usually use lower case bold symbols to represent points on the focal plane, and upper case bold symbols to represent points in space.

Projections

The usual 3-space is referred to here as *Euclidean* space. We will do these mathematical operations in a higher-dimensional space, called "projective space." Projective 4-space is

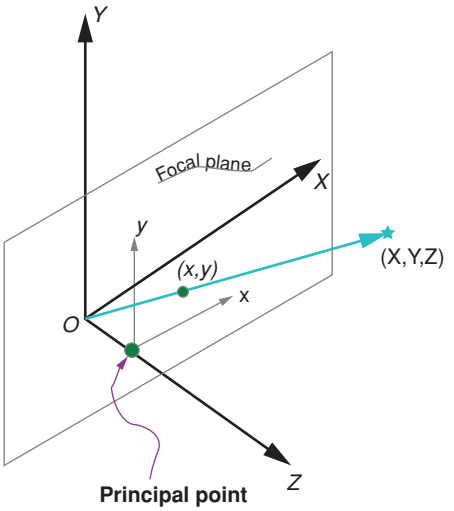


Figure 12.1 The origin of the world coordinates is at the focal point, $\mathcal{O}$. The origin of the focal plane is at the principal point. Note that the x axis of the focal plane is parallel to the X axis of the world, as is the y and Y. In this model of the projective camera, a point in space with coordinates (X, Y, Z) is projected onto a point on the image plane with coordinates (x, y). The camera origin, in turn, is related to the world origin by the projection matrix, which describes the camera, the position of the camera, and the direction of the camera.

a four-dimensional space of which conventional 3-space is a subspace. A point in 3-space relates to its corresponding point in projective 4-space by (for some scalar w)

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \Longleftrightarrow \begin{bmatrix} Xw \\ Yw \\ Zw \\ w \end{bmatrix}. \tag{12.1}$$

The mapping from 3- to 4-space is one-many, but two cases are especially interesting: If $w = 1$ we have the particularly simple case where the first three elements of the projective space representation are equal to the Euclidean representation. The other especially interesting case is the (infinite) set of points where $w = 0$. These are called "points at infinity." One of the powers of the projective representation is the fact that infinitely-distant points may be conveniently represented. The use of a 3-vector augmented by a 1 is often referred to by the term "homogeneous coordinates," as we saw in Chapter 10.

12.2.2 The Projective Camera

Most cameras can be modeled with a fairly simple model, the "projective camera," which is illustrated in Figure 12.1. Some particular point in the world, $\mathbf{X} = [X\ Y\ Z\ 1]^{\mathrm{T}}$, is projected onto the focal plane[1] by rays that pass through the imaging plane, and meet at the camera center, $\mathcal{O}$, also referred to as the *focal point*. The ray from the focal point perpendicular to

[1] We will often refer to the focal plane as the imaging plane.

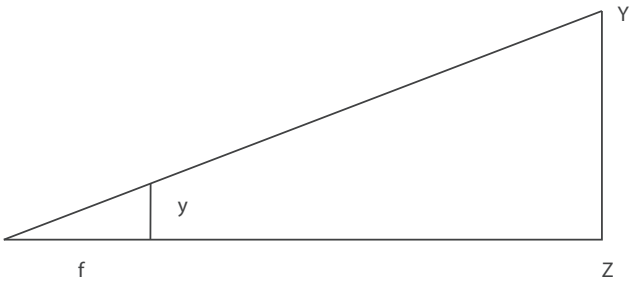


Figure 12.2 y is the height on the focal plane, and f is the distance from the focal plane to the focal point. Y is the height of the object being photographed, and Z is the distance from the camera.

the focal plane is the *principal axis*, and the principal axis intersects the focal plane at the *principal point*, which is usually chosen as the origin of the 2D camera coordinate system. This distance from the focal point to the principal point is the *focal length*, f.

The point where the ray from $\mathbf{X}$ to $\mathcal{O}$ intersects the imaging plane has coordinates $\mathbf{x} = [x\ y\ 1]$, which may be determined by knowing the focal length, f, and $\mathbf{X}$, by using similar triangles. (See Figure 12.2). In that image, we note that

$$\frac{y}{f} = \frac{Y}{Z} \Rightarrow y = \frac{fY}{Z} \quad \text{and} \tag{12.2}$$

$$\frac{x}{f} = \frac{X}{Z} \Rightarrow x = \frac{fX}{Z}. \tag{12.3}$$

We have already mentioned that in homogeneous coordinates, $\begin{bmatrix} \alpha x \\ \alpha y \\ \alpha \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$. We can now define a matrix that relates 2D to 3D, using

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} fX/Z \\ fY/Z \\ 1 \end{bmatrix} = \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{12.4}$$

Since we are using homogeneous coordinates, the vector $[fX\ fY\ Z]^{\mathrm{T}}$ may be divided by its third element to find the "real" coordinates on the image plane where the 3-space point is projected, $\mathbf{x} = [fX/Z\ fY/Z\ 1]^{\mathrm{T}}$. This leads to a simple linear relationship between points in the image and points in the world:

$$\mathbf{x} = P\mathbf{X}. \tag{12.5}$$

Equation 12.5 also defines the "Projection Matrix" P.

Some other point on the focal plane may be chosen as the origin of the imaging coordinate system. For example, many systems choose the upper left of the image as the origin and define the y axis with its positive direction down. If the origin is not at the principal point, that information may be added to the projection matrix as a translation:

$$\begin{bmatrix} f & 0 & dx & 0 \\ 0 & f & dy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{12.6}$$

It is common to define a submatrix of the projection matrix as one that involves only properties of the camera, and to call it the *camera calibration matrix*, K.

$$\mathrm{K} = \begin{bmatrix} f & 0 & dx \\ 0 & f & dy \\ 0 & 0 & 1 \end{bmatrix}. \tag{12.7}$$

It is important to emphasize that dx and dy only refer to the location of the image origin relative to the principal point.[2] They are not the $x - y$ coordinates of the camera in space. Therefore K only characterizes properties of the camera itself, not where it is located or which way it is pointing. Therefore the elements of K are often referred to as *intrinsic parameters*. The definition of K also allows us to write Eq. 12.5 in a different way:

$$\mathbf{x} = \mathrm{K}[I \mid \mathbf{0}]\mathbf{X}. \tag{12.8}$$

Using this notation, K is a 3×3 matrix. $[I \mid \mathbf{0}]$ is a single 3×4 matrix composed of a matrix, I, and a vector $\mathbf{0}$. I is the 3×3 identity matrix, and $\mathbf{0}$ is a 3×1 vector of all zeros, producing a 3×4 matrix. We say that the 3×4 matrix is produced by *concatenating* I and $\mathbf{0}$.

12.2.3 Coordinate systems

Look at Figure 12.1. Imagine you are standing on the Z axis in front of the camera, looking at the camera. From that perspective, the X axis goes off to your right, as does the x axis. The y axis goes up, and the Z axis goes right through you (not very comfortable is it?). Curl the fingers of your right hand from the X axis to the Y axis. Notice that your right thumb points in the direction of the Z axis, and you recognize that this is a right-hand coordinate system. If you were to walk behind the camera and look at these coordinates, you would see the X going off to your left, but these would STILL be the correct coordinates. Don't be confused by thinking about standing behind the camera.

Moving the Camera

The matrix $P = \mathrm{K}[I \mid \mathbf{0}]$ tells how to relate a point in space to a point on the focal plane of a camera *at the origin*. But suppose the camera moves? Suppose it both rotates in 3-space and its origin is translated? We can include that information in the projection matrix.

Before we can derive the modified projection matrix, we need to understand the issues concerning rotation of a camera. Figure 12.3 illustrates rotation of a camera about Y so the $X - Z$ plane is shown. In that figure, no translation is shown. The coordinate system $\psi_1 = [X_1, Y_1, Z_1]^{\mathrm{T}}$ is the original coordinate frame, and that system/camera is then rotated to create a second system $\psi_2 = [X_2, Y_2, Z_2]^{\mathrm{T}}$. Of course, you can't see the Y axis in that figure because it goes into the paper.

In Figure 12.3, the camera has been clearly rotated, and about the Y axis. But in which way?

[2] Remember: Right now, the origin of the camera and the origin of the world are the same. This is called *central projection*. That is about to change.

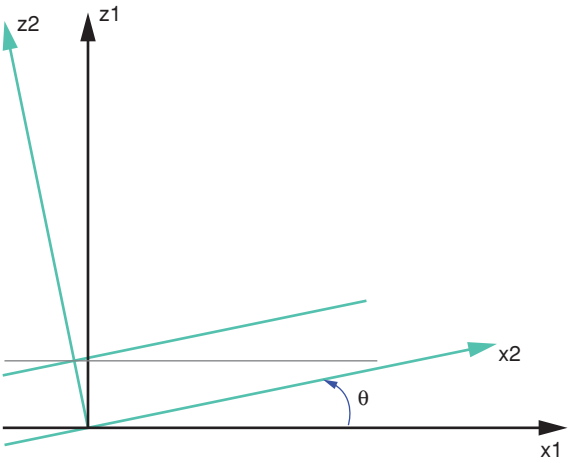


Figure 12.3 Two coordinate frames are illustrated, both of which have the same origin. The frame drawn in black is the original, denoted ψ_1 and the green, denoted ψ_2, is a result of rotation. Also illustrated are the focal planes in black and in green.

Remember the cross product, and especially recall one version of the cross product, which is $X \times Y = Z$. Application of this rule to Figure 12.3 shows that the only way to get a right-handed coordinate system is for Y to point away, into the paper. So positive Y is DOWN, depending on where the observer is located! If that is the case, then the rotation θ shown in the figure is negative, actually -20 degrees. That number will be used later, after rotation matrices are introduced.

In three dimensions, the rotation matrix[3] for positive rotation about the positive Y axis is

$$R_Y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}. \tag{12.9}$$

Constructing the Projection Matrix

Suppose our second camera is rotated by a rotation matrix $\tilde{R}$ and then located at a point $\mathbf{c}_2 \neq \mathbf{0}$. We seek a projection matrix, P_2 that will project a point in 3-space onto the focal plane of this camera. $\tilde{R}$ moves the *camera*. However, what we need is not "where is the camera?" but rather "where is the world, as viewed by this camera?" Remember, in some sense, the observer is *inside* the camera, and the world is rotating. To take this into account, all we need is a transpose:

$$R = \tilde{R}^{\mathrm{T}}. \tag{12.10}$$

With that observation, we can construct the projection matrix as follows:

1. Construct the matrix M by multiplying the rotation matrix by the concatenation of the 3×3 identity and the displacement vector. We need the version of the displacement

[3] Look in Chapter 3 for the other rotation matrices and properties of rotation matrices.

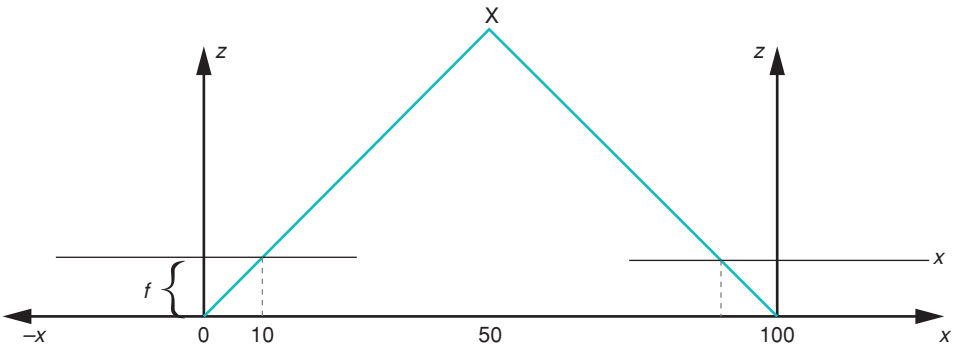


Figure 12.4 Two cameras viewing the same object. Neither camera is rotated. Viewed with the positive Y axis into the paper.

vector that describes the origin in the coordinates of the camera, (not the camera in terms of the origin) $-\mathbf{c}_2$, forming a 3×4 matrix denoted by M.

$$M = R\,[I|\,-\mathbf{c}_2]\,.$$

M contains information about the position and orientation of Camera 2. Its elements are called the *extrinsic parameters* of the camera.

2. Finally, the projection matrix is constructed by premultiplying by the matrix of intrinsic parameters.

$$P_{3\times4} = K_{3\times3}M_{3\times4} \tag{12.11}$$

where the matrix dimensions are shown for clarity.

Now any point in the view of the camera $\mathbf{X}$ projects to a point on the camera focal plane $\mathbf{x}$ obeying (writing Eq. 12.5 again):

$$\mathbf{x} = P\mathbf{X} \tag{12.12}$$

Example with no Rotation

In Figure 12.4 two cameras view the same object, both of which have the same focal length, 10, and no other internal distortions. Thus for both cameras, the intrinsic matrix is

$$K = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Suppose camera 1 is at the origin ($\mathbf{c}_1 = \mathbf{0}$), and pointed along the Z axis of the global coordinate system. Then its projection matrix is

$$P_1 = K \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Given a point in 3-space $X = [\,50 \quad 0 \quad 50 \quad 1\,]^{\mathrm{T}}$, we operate on it by multiplying

$$\mathbf{x}_1 = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 50 \\ 0 \\ 50 \\ 1 \end{bmatrix} = \begin{bmatrix} 500 \\ 0 \\ 50 \end{bmatrix},$$

remembering that this homogeneous 3 vector actually represents a point in a 2-space, the camera plane. We divide by 50, and find that the point on the focal plane is $\mathbf{x}_1 = [\,10 \quad 0 \quad 1\,]^{\mathrm{T}}$.

Let us locate the second camera at $\mathbf{c}_2 = [\,100\ 0\ 0\,]^{\mathrm{T}}$ relative to the first camera, still pointing in the same direction. The second projection matrix is found by constructing a matrix

$$P_2 = KR_2[I| - \mathbf{c}_2]. \tag{12.13}$$

Since there is no rotation here, $R_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. Substituting into Eq. 12.13,

$$P_2 = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -100 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{12.14}$$

and

$$P_2 = \begin{bmatrix} 10 & 0 & 0 & -1000 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{12.15}$$

and applying this matrix to the same point $X = [\,50 \quad 0 \quad 50 \quad 1\,]^{\mathrm{T}}$, we find X appears on the focal plane of camera 2 at $\mathbf{x}_2 = [\,-10 \quad 0 \quad 1\,]^{\mathrm{T}}$.

This example is illustrated in Figure 12.4. The horizontal axis of the figure is labeled with world coordinates. However, the values returned by the projection matrices are on the focal planes, relative to the principal point of each camera. The ray from the object X to camera 2 penetrates the focal plane of camera 2 at $x = -10$, not $X = 90$.

Example with Rotation of the Second Camera

As discussed above, the second camera is to be rotated about the positive Y axis through an angle θ, a rotation described by

$$\tilde{R} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}. \tag{12.16}$$

$\tilde{R}$ is the rotation matrix that moves camera 2. To see how camera 2 sees the world, we need the transpose of $\tilde{R}$, denoted just R.

$$R = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}. \tag{12.17}$$

Now we have the proper version of R.

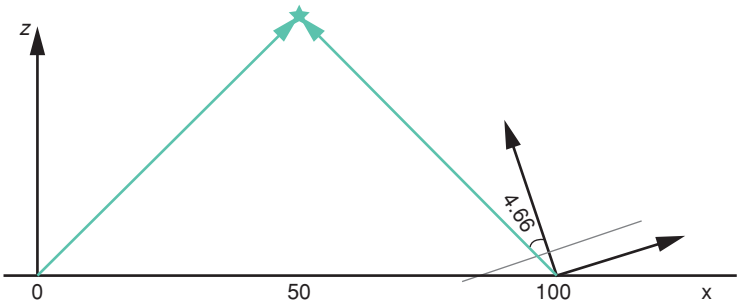


Figure 12.5 Top view of two cameras. The camera on the right is rotated by -20 degrees about the positive y axis (which goes into the paper). The point where the ray intersects the focal plane is $x = -4.6$ on the right camera and $x = 10$ on the left camera.

In the following, for notational convenience, $\sin\theta$ and $\cos\theta$ are replaced by s and c, respectively.

Now we need to concatenate the identity matrix with the vector to the origin, as before, and premultiply by K:

$$P_2 = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{bmatrix} \left[\begin{array}{ccc|c} 1 & 0 & 0 & -100 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right] \tag{12.18}$$

then

$$P_2 = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c & 0 & -s & -100c \\ 0 & 1 & 0 & 0 \\ s & 0 & c & -100s \end{bmatrix}.$$

$$P_2 = \begin{bmatrix} 10c & 0 & -10s & -1000c \\ 0 & 10 & 0 & 0 \\ s & 0 & c & -100s \end{bmatrix}$$

Multiplying this matrix by the point X that we have used before, we find

$$\mathbf{x} = \begin{bmatrix} 10c & 0 & -10s & -1000c \\ 0 & 10 & 0 & 0 \\ s & 0 & c & -100s \end{bmatrix} \begin{bmatrix} 50 \\ 0 \\ 50 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 500c - 500s - 1000c \\ 0 \\ 50s + 50c - 100s \end{bmatrix} = \begin{bmatrix} -500(s + c) \\ 0 \\ 50(c - s) \end{bmatrix}.$$

Now, take a look at Figure 12.5.

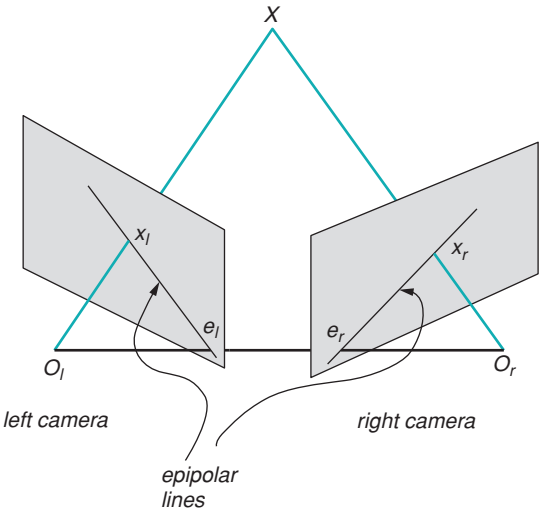


Figure 12.6 Two cameras view the same point, X. In the right-hand image, this point is seen at point $\mathbf{x}_r$, and at $\mathbf{x}_l$ in the left-hand image. All the 3D points along the line $\mathcal{O}_l X$ project to the same image point, $\mathbf{x}_l$ in the left image.

In that figure,

$$\begin{aligned} c &= \cos(-20^o) = 0.939 \\ s &= \sin(-20^o) = -0.342 \\ c - s &= 1.281 \\ c + s &= 0.597. \end{aligned}$$

Substituting these values into the right-hand side of Eq. 12.19, we find

$$\mathbf{x}_2 = \begin{bmatrix} -500(s+c) \\ 0 \\ 50(c-s) \end{bmatrix} = \begin{bmatrix} -298.5 \\ 0 \\ 64.05 \end{bmatrix}.$$

Dividing by the third element in the homogeneous vector, produces

$$\mathbf{x}_2 = \begin{bmatrix} -4.66 \\ 0 \\ 1 \end{bmatrix}.$$

Now we see that there is a definite relationship between points in space and points on the focal plane. It should be easy now to figure out how to do stereopsis[4]: just take two corresponding points and find the point in 3-space where the rays from the cameras intersect.

Ah, but wait!

In Figure 12.6, a line is shown from the point $\mathcal{O}_r$ to point x_r, and continued toward the 3-space point X. Similarly, the line $(\mathcal{O}_l, x_l)$ is also continued toward X. But remember,

[4] But of course, it's not easy.

these are lines in 3-space, and unlike lines on the plane, they don't have to intersect; in fact, they almost never intersect. Furthermore, our "magic" correspondence finder told us that the neighborhood of x_l looks somewhat like the neighborhood of x_r. That is true, not all proposed correspondences are correct. In the following section, the correspondence problem will be solved without needing the camera matrices, and allowing for a few, small errors.

12.3 Shape from Motion – Range from Two Unknown Cameras

In the previous section, we knew where the cameras were and how they were pointed. In this section, we relax those requirements, and replace them with the requirement for a reasonable number of good correspondences.

Notice that most mammals have two eyes, and therefore, one would suspect they are using that dual information to infer something about the three-dimensional world. This suspicion is reinforced by the work of Julesz, who showed [12.20] that random dot stereograms could be used to construct an image that looked totally like random noise unless viewed from just the right distance, but with proper viewing, the same images appear to have depth.

The trick to such perception makes use of *correspondence* between the two eyes; that is, that an area appears so similar to both eyes that both may consider it the same. Given a correspondence, the brain can compute an estimate of the 3D position of that area. We pursue that idea rigorously below.

12.3.1 Stereopsis and the Correspondence Problem

Suppose we have two cameras viewing the same scene and we do not know the camera poses.[5] We do, however, have an operator that is capable of saying "this point in image 1 looks like that point in image 2." That is, we have an operator that can solve the correspondence problem, but not always correctly.

As in the previous section, homogeneous coordinates are used to approach this problem. That is, the 2-vector $[x\ y]^{\mathrm{T}}$ is represented by the 3-vector $[x\ y\ 1]$. Furthermore, we will be dealing with lines in the plane with parameters a, b and c, such that the equation for a line becomes

$$\mathbf{l}^{\mathrm{T}}\mathbf{x} = 0 \tag{12.19}$$

where $\mathbf{l} = [a\ \ b\ \ c]^{\mathrm{T}}$ and $\mathbf{x} = [x,\ y,\ \ 1]^{\mathrm{T}}$.

Now here is an interesting fact: if point $\mathbf{x}_l$ in the left-hand image and point $\mathbf{x}_r$ in the right-hand image really do correspond, then there exists a matrix F that satisfies

$$\mathbf{x}_l^{\mathrm{T}} F \mathbf{x}_r = 0. \tag{12.20}$$

Furthermore, if there are a lot of points in the left-hand image $\mathbf{x}_{li}$, and a lot of corresponding points $\mathbf{x}_{ri}$ in the right-hand image, the *same* matrix F satisfies $\mathbf{x}_{li} F \mathbf{x}_{ri} = 0$, for all i. The

[5] "Pose" is a set of 6 numbers, including position (x, y, z) and orientation (roll, pitch, yaw) in 3-space.

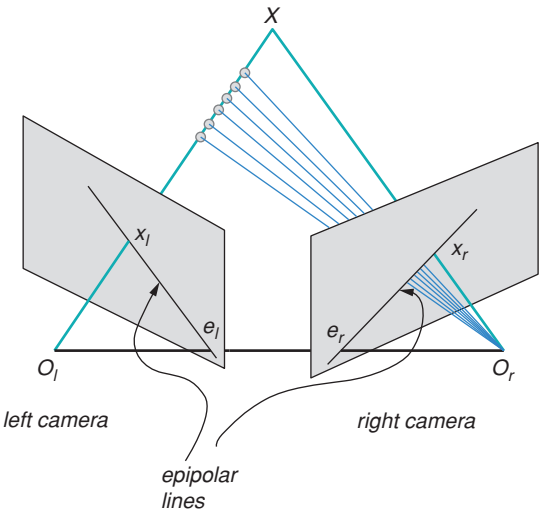


Figure 12.7 We know the point X must lie along the ray through x_l, but from just the left image we don't know where. In the right image, however, the ray projects to a single line, the epipolar line.

correspondence problem then seems easy: if we think two points might correspond, then substitute them into Eq. 12.20. If the result is zero then the two points correspond.

We hope you have caught the challenge in the previous paragraph: We don't know F *and* we don't know which correspondences are correct.

As illustrated in Figure 12.6, a single point in one image can correspond to any of a set of points, which lie on a straight line, in the other image. The line is referred to as the *epipolar line* of $\mathbf{x}$, as illustrated in Figure 12.7.

F is called the *fundamental matrix*, and it may be estimated using the "eight point algorithm" [12.1], which will be briefly discussed in the next section. However, there is a problem: It takes 8 correspondences to estimate F. If we choose 8 at random, there is a fair likelihood that one of them is wrong. If one estimate is wrong, then F is probably wrong, so we can't use it to test which correspondences are true. It sounds like we are doomed, doesn't it?

Following is one approach: Start by finding a large (significantly more than 8) set of possible correspondences, (remember, a correspondence is two ordered triples of coordinates, a triple $\mathbf{x}_l = [x_l, y_l, 1]^{\mathrm{T}}$ from the left image and a triple $\mathbf{x}_r = [x_r, y_r, 1]^{\mathrm{T}}$ from the right image). Denote the set of possible correspondences S_c.

1. From S_c choose 8 possible correspondences at random.
2. From those 8 correspondences, calculate an estimate of F. (See section 12.3.2.)
3. Identify the *inliers*. Inliers are correspondences that come very close to satisfying Eq. 12.20. We determine them by the following:
   (a) Find both epipolar lines. The epipolar line corresponding to $\mathbf{x}_r$ we denote as $\mathbf{l}_l$ and the epipolar line corresponding to $\mathbf{x}_l$ we denote as $\mathbf{l}_r$. Remember from Eq. 12.19 that a line is a 3-vector. We can easily find the epipolar lines using

$$\mathbf{l}_l = F\mathbf{x}_r \tag{12.21}$$

$$\mathbf{l}_r = F\mathbf{x}_l. \tag{12.22}$$

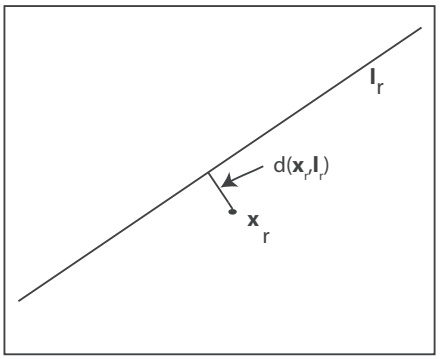


Figure 12.8 The right-hand image of a pair of images containing one pair of possibly corresponding points: $\mathbf{x}_l$ in the left image and $\mathbf{x}_r$ in the right. The line $\mathbf{l}_r$ is the set of all points in the right image that could correspond to $\mathbf{x}_l$. If the correspondence were perfect, $\mathbf{x}_r$ would lie on this line. Therefore the distance $d(\mathbf{x}_l, \mathbf{l}_r)$ is a measure of the possible error in this proposed correspondence

(b) Find the *epipolar line distances*, illustrated in Figure 12.8. This distance may be found by $d(\mathbf{x}_l, \mathbf{l}_l) = \frac{|\mathbf{x}_l^T \mathbf{l}_l|}{(a_l^2 + b_l^2)^{1/2}}$ in the left image, and similarly in the right. (Remember that $\mathbf{l}$ is the vector $[a \;\; b \;\; c]^{\mathrm{T}}$, the coefficients of the equation $ax + by + c = 0$).

(c) Compute an error

$$E = d(\mathbf{x}_l, \mathbf{l}_l) + d(\mathbf{x}_r, \mathbf{l}_r). \tag{12.23}$$

(d) If E is too large, it is an outlier, otherwise it is an inlier. Therefore a threshold will be needed. Many applications choose a maximum distance of 1 pixel.

4. At iteration k, the estimate of F just found in the previous step is denoted F_k. Test this value of F over all the proposed correspondences in S_c and keep track of all the correspondences that are consistent with F_k. We will call this the *inlier set* that has cardinality $|S_k|$. If $|S_k|$ is the largest seen so far, store the fact that F_k is the best seen so far. That is, F_k explains more of the inlier correspondences.
5. Randomly choose another 8 points and find another possible F, repeating the steps above until some maximum, N, iterations have been run, or we have found a single F that has all the correspondences in its inlier set.

The algorithm above is a particular version of a very general method, Random Sample Consensus (RANSAC) [12.8, 12.52] for finding models in the presence of noise that may contain outliers. It is perhaps easiest to see the concept of an outlier in the context of determining if a point is from a particular Gaussian probability distribution: if the point is more than three standard deviations away from the mean, we would typically call that point an outlier. In this application, we do not use a probabilistic model, but the philosophy is the same.

And yes, we still haven't told you how to find the 3D position in space.

12.3.2 The 8-Point Algorithm

Before we continue, we should discuss the range of the numbers being dealt with. Using pixel coordinates, typical images have coordinates between 0 and around 1000. Algorithms

must therefore deal with arithmetic operations involving numbers ranging over three orders of magnitude. In solving such a system, double precision is likely to be required, and the resulting numbers may still be huge. (Consider computing a determinant of a 20×20 matrix of numbers between 0 and 1000. Intermediate results involving numbers on the order of 1000^{20} would not be impossible). The following normalization [12.28] provides significant improvement:

- Move the origin of the coordinate system to the center of the image.
- Scale the coordinates so that the average distance of a point to the origin is $\sqrt{2}$.

This technique can be critically important. It is sometimes referred to as *preconditioning* the data.

There are many variations of the 8-point algorithm. In this section, we describe a simple version, so that the student may grasp the basic principles. Of course, this won't be the absolute best version, but we will also provide pointers to the literature for the student who needs to implement these concepts in robust software.

Start by rewriting Eq. 12.20:

$$[x_l \;\; y_l \;\; 1] \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix}. \tag{12.24}$$

Multiplying the matrices out, we get

$$x_l x_r f_{11} + y_l x_r f_{21} + x_r f_{31} + x_l y_r f_{12} + y_l y_r f_{22} + y_r f_{32} + x_l f_{13} + y_l f_{23} + f_{33} = 0, \tag{12.25}$$

which we can reformulate by defining

$$\mathbf{a} = [x_l x_r \; y_l x_r \; x_r \; x_l y_r \; y_l y_r \; y_r \; x_l \; y_l \; 1]^{\mathrm{T}} \tag{12.26}$$

and

$$\mathbf{f} = [f_{11} \; f_{21} \; f_{31} \; f_{12} \; f_{22} \; f_{32} \; f_{13} \; f_{23} \; f_{33} \;]^{\mathrm{T}} \tag{12.27}$$

which converts Eq. 12.24 into

$$\mathbf{a}^{\mathrm{T}}\mathbf{f} = 0. \tag{12.28}$$

Equation 12.24, rewritten into Eq. 12.28, only represents a single correspondence. If we have some number of such equations, we can stack them as shown below

$$\begin{aligned} \mathbf{a}_1^{\mathrm{T}}\mathbf{f} &= 0 \\ \mathbf{a}_2^{\mathrm{T}}\mathbf{f} &= 0 \\ &\cdots \\ \mathbf{a}_8^{\mathrm{T}}\mathbf{f} &= 0 \end{aligned} \tag{12.29}$$

and rewrite this system with a single matrix equation:

$$A\mathbf{f} = \mathbf{0}. \tag{12.30}$$

Equation 12.30 may be solved for **f** using singular valued decomposition (SVD) (see section 3.2.9). SVD will find three matrices, U, D, and V, such that $A = UDV^{\mathrm{T}}$. The column of V corresponding to the smallest singular value will be the least-squares estimate of **f**.

F is then reconstructed from **f** using Eq. 12.27.

One problem remains: it is very important that F be singular; specifically of rank 2. Again use SVD, writing $F = UDV^{\mathrm{T}}$, and D will be diagonal with the three singular values on the diagonal,

$$F = U \begin{bmatrix} D_{11} & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & D_{33} \end{bmatrix} V^{\mathrm{T}}.$$

Depending on the implementation, the smallest singular value will be either D_{11} or D_{33}. We need to set the smallest one to zero. Assume for now that the smallest singular value is D_{33}.

The rank 2 diagonal matrix closest D will be

$$D' = \begin{bmatrix} D_{11} & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

The new estimate of F is then constructed from

$$F = UD'V^{\mathrm{T}} \tag{12.31}$$

and this estimate will have rank 2, as required.

So, given correspondences, it is possible to eliminate erroneous correspondences and find the fundamental matrix. In the next section, we find the camera matrices from F.

12.3.3 Finding the Camera Matrices

You could try getting out your tape measure and your protractor and measuring the position and orientation of the two cameras. In principle, you could find the camera matrices in that way. But, let's think about that: Let's say your digital camera has a 1000 × 1000 array of pixels on a 1cm × 1cm chip, which means the pixels are spaced ten microns apart. A measurement error of ten microns displaces your camera origin by one pixel. Exactly how accurate IS your tape measure? Instead of measuring, we can use the camera itself to figure out the camera matrices.

In section 12.3.1 you learned how to find the fundamental matrix. This subsection presents one way to find the two camera matrices from the fundamental matrix.

Given a fundamental matrix F, the two camera matrices may be chosen to be

$$P_l = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \text{ and } P_r = [[\mathbf{e}']_{\times} F \mid \mathbf{e}']. \tag{12.32}$$

Here $\mathbf{e}'$ is the epipole satisfying $\mathbf{e}'^{T} F = 0$.

To find this epipole, we just observe that $\mathbf{e}'$ is in the left null space of F. You might think this is a problem until you realize that F has rank 2. Thus, when you find the null space of F^{T} (see section 3.2.3) and use the observation that the third element of $\mathbf{e}'$ is equal to one, you realize that there is just one element in the null space, except of course, for the zero vector and all scalar multiples of $\mathbf{e}'$. Therefore we can find $\mathbf{e}'$ from the left null space of F, and we can find the camera matrices from that.

12.3.4 Stereopsis from Camera Matrices

Given two cameras with camera matrices P_l and P_r, those matrices provide two vector equations:

$$\mathbf{x}_l = P_l\mathbf{X}, \quad \mathbf{x}_r = P_r\mathbf{X}.$$

Writing either of these equations out for some matrix P,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{12.33}$$

Recall from Eq. 3.3 that the cross product of a vector with itself is zero. We use this fact, observing that $\mathbf{x} = P\mathbf{X}$ implies $\mathbf{x} \times P\mathbf{X} = 0$, where $\times$ is the cross product.

Define $\mathbf{P}_i$ to be the i^{th} row of P. This will be a 3-element *row* vector. In the next equation, an inner product will be performed between each of the rows of the P matrix with $\mathbf{X}$. Since these vectors are row vectors, no transpose will be required.

The cross product produces

$$\begin{aligned} [y\mathbf{P}_3 - \mathbf{P}_2]\,\mathbf{X} &= 0 \\ [-(x\mathbf{P}_3 - \mathbf{P}_1)]\,\mathbf{X} &= 0 \\ [x\mathbf{P}_2 - y\mathbf{P}_1]\,\mathbf{X} &= 0. \end{aligned} \tag{12.34}$$

To assist the student in understanding the notation, we expand the first line of Eq. (12.34)

$$(yP_{31} - P_{21})X + (yP_{32} - P_{22})Y + (yP_{33} - P_{23})Z + yP_{34} - P_{24} = 0.$$

Because it is equal to zero, we may reverse the sign of the middle term of Eq. 12.34.

Consider the linear independence of the three equations in Eq. (12.34). Multiply the first equation by x, the second by y, and add them. Since that produces the third equation, we recognize that these are not linearly independent. Since it provides no additional information, drop the third equation, leaving the two equations:

$$\begin{aligned} [y\mathbf{P}_3 - \mathbf{P}_2]\mathbf{X} &= 0 \\ [x\mathbf{P}_3 - \mathbf{P}_1]\,\mathbf{X} &= 0. \end{aligned} \tag{12.35}$$

The derivation above referred to an arbitrary matrix $\mathbf{P}$. To do stereopsis, we use Eq. 12.35 substituting P_l for P, and again for P_r, to produce a system of four equations:

$$\begin{aligned}
[y\mathbf{P}_{l3} - \mathbf{P}_{l2}]\,\mathbf{X} &= 0 \\
[x\mathbf{P}_{l3} - \mathbf{P}_{l1}]\,\mathbf{X} &= 0 \\
[y\mathbf{P}_{r3} - \mathbf{P}_{r2}]\,\mathbf{X} &= 0 \\
[x\mathbf{P}_{r3} - \mathbf{P}_{r1}]\,\mathbf{X} &= 0
\end{aligned} \tag{12.36}$$

Now we have four equations in three unknowns, and Eq. 12.36 may be rewritten as a matrix-vector product:

$$A\mathbf{X} = 0. \tag{12.37}$$

$\mathbf{X}$ may be found ([12.15], page 90) as the eigenvector corresponding to the smallest eigenvalue of $A^{\mathrm{T}}A$.

12.3.5 A Fundamental Ambiguity

Everything we have done in this chapter since section 12.3 has been based on the observation that it is difficult to measure camera pose to the accuracy one would like. So we have used correspondences to find both the projection matrices and the 3D spatial points. We haven't actually measured *anything*. To see where this could get us in trouble, think about the following rather basic equation that you have seen before:

$$\mathbf{x} = P\mathbf{X}. \tag{12.38}$$

By finding correspondences, we found a matrix P, and, for many of the interesting points in the image, $\mathbf{x}_i$, we have found the corresponding spatial point $\mathbf{X}_i$. We can write Eq. 12.38 as

$$\mathbf{x} = PI\mathbf{X}, \tag{12.39}$$

where I is the 4×4 identity matrix.

Now let H be any invertible 4×4 matrix. We can replace I by HH^{-1}, producing

$$\mathbf{x} = PHH^{-1}\mathbf{X} = (PH)(H^{-1}\mathbf{X}). \tag{12.40}$$

We now notice that PH can be thought of as a camera matrix, and $H^{-1}\mathbf{X}$ can be thought of as a spatial point. These *also* satisfy the conditions of Eq. 12.38.

There are really an infinite number of matrix/point pairs that can satisfy the conditions. To truly solve the shape from motion problem, we need *some* actual measurement; otherwise, we cannot tell the difference between a toy tank up close and a real tank further away.

There are several approaches to solving this *triangulation* problem. The methods of solution already discussed are appropriate to this problem as well. The reader is referred to [12.15] for additional detail.

12.4 Image Stitching and Homographies

This section discusses *homographies* that map between points in two images. The coordinate representation used is *homogeneous* coordinates. That is, let a point in image 1 be $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$, and suppose the corresponding point in image 2 is $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$. Then the 3×3 matrix H will have the property that

$$\begin{bmatrix} w'x' \\ w'y' \\ w' \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{12.41}$$

From $\begin{bmatrix} w'x' \\ w'y' \\ w' \end{bmatrix}$, one can find the point on the focal plane by dividing by the third coordinate, w'.

Expanding the previous equation, we find

$$\begin{bmatrix} wx' \\ wy' \\ w' \end{bmatrix} = \begin{bmatrix} h_{11}x + h_{12}y + h_{13}w \\ h_{21}x + h_{22}y + h_{23}w \\ h_{31}x + h_{32}y + h_{33}w \end{bmatrix}. \tag{12.42}$$

Since these are homogeneous representations of vectors, they represent projective operations.

As you follow the derivation of H, you will find that the process of finding homographies is similar to that of finding the fundamental matrix. Both are based on finding correspondences, and make use of the cross product. Rather than simply pointing out the differences from finding F, the derivation for finding H is presented in complete form here.

The algorithm needs as many correspondences as possible (the more you find, the better it will work . . . at least ten) between the two pictures. Obviously these must be in the overlapped area of the two images.

Given a correspondence, $\mathbf{x}' = H\mathbf{x}$, we make use of the same observation that we did in section 12.3.4, that if two vectors are equal, their cross product is zero.

Given two vectors, $\mathbf{u}$ and $\mathbf{v}$, the first element of the cross product is $u_2v_3 - u_3v_2$, and in this problem

$$\mathbf{u} = \begin{bmatrix} wx' \\ wy' \\ w' \end{bmatrix} \quad \mathbf{v} = \begin{bmatrix} h_{11}x + h_{12}y + h_{13}w \\ h_{21}x + h_{22}y + h_{23}w \\ h_{31}x + h_{32}y + h_{33}w \end{bmatrix} \tag{12.43}$$

and the first element of the cross product becomes

$$c_1 = u_2v_3 - u_3v_2 = y'(h_{31}x + h_{32}y + h_{33}w) - w'(h_{21}x + h_{22}y + h_{23}w) \tag{12.44}$$

Since we need to write H as a vector, $\mathbf{h}$, define $h_1 = h_{11}, h_2 = h_{12}, h_3 = h_{13}, h_4 = h_{21}, \ldots h_9 = h_{33}$, and then sort the terms of Eq. 12.44 by the subscript on h, so that we

get the equation

$$c_1 = u_2 v_3 - u_3 v_2 = 0 + 0 + 0 - w'xh_4 - w'yh_5 - w'wh_6 + y'xh_7 + y'yh_8 + y'wh_9. \tag{12.45}$$

Once written in this form, we recognize that the first element of the cross product can be written as an inner product, and setting that equal to zero, we get an equation,

$$c_1 = [0\ 0\ 0\ -w'x\ -w'y\ -w'w\ y'x\ y'y\ y'w] \cdot \mathbf{h} = 0. \tag{12.46}$$

However, we don't actually have nine degrees of freedom in this problem, since the same equation will hold true no matter how we scale the images. This leaves us free to choose a value for one element of $\mathbf{h}$. The usual convention is to choose $h_9 = 1$. Doing so gives us a constant value to put on the right-hand side, producing

$$[0\ 0\ 0\ -w'x\ -w'y\ -w'w\ y'x\ y'y] \cdot \mathbf{h} = -y'w. \tag{12.47}$$

Similarly, the second element of the cross product produces an equation

$$[xw'\ yw'\ ww'\ 0\ 0\ 0\ -xx'\ -yx'] \cdot \mathbf{h} = -wx'.$$

From these two equations, we now develop an algorithm:

For each correspondence, $x_i, y_i \Longleftrightarrow x'_i, y'_i$ construct this 2×8 matrix.

$$\begin{bmatrix} 0 & 0 & 0 & -x_i & -y_i & -1 & x_i y'_i & y_i y'_i \\ x_i & y_i & 1 & 0 & 0 & 0 & -x_i x'_i & -y_i x'_i \end{bmatrix}.$$

Supposing that you found n correspondences, stack the 2×8 matrices to form a $2n \times 8$ matrix. Call that matrix A.

For each correspondence, construct the 2×1 vector

$$\begin{bmatrix} -y'_i \\ x'_i \end{bmatrix}.$$

Stack these vectors to product a $2n \times 1$ vector named $\mathbf{D}$, satisfying

$$A^{\mathrm{T}}\mathbf{h} = \mathbf{D}. \tag{12.48}$$

Compute the vector $\mathbf{h}$ using

$$\mathbf{h} = (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}D.$$

The previous equation is rewritten explicitly showing matrix/vector sizes:

$$\mathbf{h}_{8\times 1} = \left(A^{\mathrm{T}}_{8\times 2n} A_{2n\times 8}\right)^{-1}_{8\times 8} A^{\mathrm{T}}_{8\times 2n} D_{2n\times 1}.$$

Note that this requires inversion of an 8×8 matrix.

Figure 12.9 Two images, taken from the same point, but with rotation about the Y axis, and slight rotation about the Z axis. The authors are grateful to JunYi Liu for this figure.

You now have an 8-element vector

$$\mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \end{bmatrix},$$

Construct the 3×3 matrix

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1.0 \end{bmatrix}.$$

The matrix H satisfies $\mathbf{x}' = H\mathbf{x}$. Thus, for any point $\mathbf{x}$ in image i_l, $H\mathbf{x}$ will be the corresponding point in image i_r.

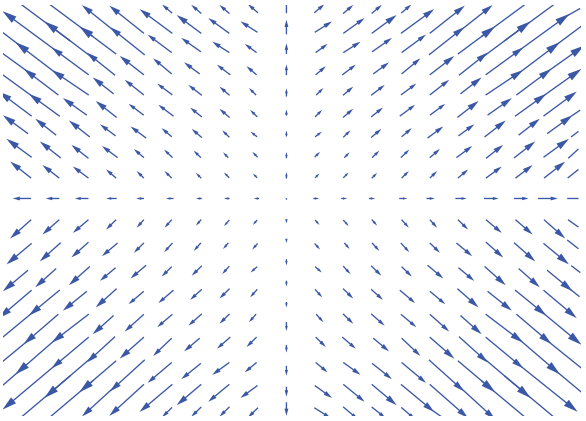
Once you have H, you can copy an image into the coordinates of another image. So for every point in I_l, find its coordinates in I_r and copy the pixel from I_l to I_r. Note that you will need to make I_r bigger since many of the points of I_l will fall outside of the original I_r.

As shown in the right-hand image, you are very likely have the same problem with black dots (which we call "interpolating in reverse") here that you had in a previous homework. If this is a problem, instead of copying from I_l to I_r using H, that is, scanning over the input image, and figuring out where it goes in the output, instead copy from I_r to I_l using H^{-1} scanning over the output image, and figuring out where it came from in the input.

If your two images fit together reasonably well, congratulations! You now can do image stitching.

An example of image stitching is presented in Figures 12.9 and 12.10.

Figure 12.10 (ABOVE) Automatic stitching of the two images in Figure 12.9. The right-hand half of the stitching may appear slightly lighter than the left. This is due to the automatic gain control (AGC) on the camera. The camera sets its gain by the average brightness in the image. For this reason, we recommend that anyone doing Computer Vision work use a camera that allows the user to disable AGC. (BELOW) Stitching of the same images, but interpolating in reverse, which causes the artifact at the right. The authors are grateful to JunYi Liu for these images.



What does this motion denote? Yes, it is camera motion, but what is happening?

Figure 12.11 Disparity between two images. Resulting from camera motion.

12.4.1 Disparity

Suppose, we have just one camera, but something in the scene moves, or, that single camera moves. From the information in the two images, we might identify the moving object, track it over time, and possibly recognize it. If the camera itself is moving, then it is possible to predict time-to-collision or camera motion. Begin with the case of a stationary camera and a single moving object:

Assume that a solution to the correspondence problem exists for this data. Collect two images, f_1, and f_2, which were taken at different times. At each pixel $\mathbf{x}_2 = [x_2, y_2]^{\mathrm{T}}$ in the second image, determine the corresponding point, $\mathbf{x}_1 = [x_1, y_1]^{\mathrm{T}}$ in the first image, if it has one. Then, the *disparity vector* is $\mathbf{x}_2 - \mathbf{x}_1$. If a set of correspondences is found, and the disparity vectors drawn as arrows, one might get an image like Figure 12.11.

In Figure 12.11, the disparity resulting from forward camera motion is observed. If sufficient correspondences are present, disparity fields like these can be analyzed to determine motion of the camera.

A special case of disparity occurs when two identical cameras are aligned to have parallel axes, as illustrated in Figure 12.4. In this case, the only motion is horizontal displacement, and that is a scalar. That scalar, in turn may be viewed as a brightness in a *disparity image* in which brighter things are closer. Figure 12.12 illustrates this. Similar results are obtained by moving the camera in a straight line perpendicular to the camera axis.

12.4.2 Matching Geometric Invariants

This section presents a special case of matching, one that demonstrates the power of the determinant, an operation that has a remarkable degree of invariance to image motions. We illustrate the power of determinants and their invariant properties.

An *invariant* is a property, measurement, or operation whose output does not change if the input undergoes a certain class of transformation. As an example, consider a matrix

Figure 12.12 Two images, taken after a motion. As objects are further away, they move less in the image, so the horizontal disparity may be converted into distance, as illustrated in the range image at the bottom. Library names: left_10.png, right_10.png

with three columns, each of which is a homogeneous coordinate in 2-space:

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix}.$$

The matrix M has determinant $|M| = 5.0$.

Now, rotate all three points about the z axis by rotation of $\pi/4$ and then translate all the points by (2,4) using a homogeneous transform matrix:

$$M' = \begin{bmatrix} 0.7071 & -0.7071 & 2 \\ 0.7071 & 0.7071 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix},$$

and the determinant of M' will be 5.0, just as it was before the motion. This is a property of determinants, and demonstrates one example in which the determinant is invariant to isometries.

Clearly, relationships exist between measurements in the 2D image and points in 3D space, and we should seek such relationships that are invariants.

This section introduces the concept of invariants using the work of Weiss and Ray [12.56] that relates invariants in three dimensions to invariants in two dimensions.

We start with simply finding a set of numbers that are invariant. The approach will be to find five points in the 3D model and calculate from them some properties that uniquely characterize them in an invariant way. Then, we will find five points in the image and determine which model they best match.

Choose a set of five feature points $\{\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4, \mathbf{X}_5\}$ from the 3D model, at least four of which are noncoplanar. It only requires four linearly independent vectors to span a 4-space.[6] Therefore, since five points cannot be linearly independent, one of them can be written as a linear combination of the others. Choose point 5.

$$\mathbf{X}_5 = a\mathbf{X}_1 + b\mathbf{X}_2 + c\mathbf{X}_3 + d\mathbf{X}_4. \tag{12.49}$$

For each of the five points, we construct matrices with four columns, using the points as columns, and using as a subscript, the index of the point we omitted. Since the determinant of a matrix of points is invariant to rigid body motions[7] we will use determinants, and write the determinant that is constructed from any four of the five points, using the same subscript notation:

$$M_1 = |\mathbf{X}_2\mathbf{X}_3\mathbf{X}_4\mathbf{X}_5|. \tag{12.50}$$

Replace X_5 with its expansion from Eq. 12.49.

$$M_1 = |X_2X_3X_4(aX_1 + bX_2 + cX_3 + dX_4|. \tag{12.51}$$

Making use of a frequently-forgotten property of determinants called the "distributive rule for columns," and the fact that multiplying a single column of a matrix by a scalar multiplies the determinant by the same amount, we substitute the form for $\mathbf{X}_5$ from Eq. 12.49 into Eq. 12.50, deriving

$$M_1 = a|\mathbf{X}_2\mathbf{X}_3\mathbf{X}_4\mathbf{X}_1| + b|\mathbf{X}_2\mathbf{X}_3\mathbf{X}_4\mathbf{X}_2| + c|\mathbf{X}_2\mathbf{X}_3\mathbf{X}_4\mathbf{X}_3| + d|\mathbf{X}_2\mathbf{X}_3\mathbf{X}_4\mathbf{X}_4|. \tag{12.52}$$

This can be simplified by observing that the determinant of a matrix that has two identical columns is zero:

$$M_1 = a|\mathbf{X}_2\mathbf{X}_3\mathbf{X}_4\mathbf{X}_1|. \tag{12.53}$$

This can be simplified even more by observing that if you interchange two columns, you flip the sign of the determinant.

$$M_1 = (-a)|\mathbf{X}_1\mathbf{X}_3\mathbf{X}_4\mathbf{X}_2| = (a)|\mathbf{X}_1\mathbf{X}_3\mathbf{X}_2\mathbf{X}_4| = (-a)|\mathbf{X}_1\mathbf{X}_2\mathbf{X}_3\mathbf{X}_4|. \tag{12.54}$$

So

$$M_1 = -aM_5. \tag{12.55}$$

Similarly

$$M_2 = bM_5 \tag{12.56}$$

$$M_3 = -cM_5 \tag{12.57}$$

$$M_4 = dM_5. \tag{12.58}$$

From this, we can write an expression for the coefficients in Eq. 12.49:

$$a = -\frac{M_1}{M_5} \quad b = \frac{M_2}{M_5} \quad c = -\frac{M_3}{M_5} \quad d = \frac{M_4}{M_5}. \tag{12.59}$$

[6] Review spanning a space in section 10.3.1.
[7] In fact, absolute invariants of linear forms are always ratios of powers of determinants [12.14].

In 2D, the same five points project to a set of 3-vectors (again, using homogeneous coordinates), and

$$\mathbf{x}_5 = a\mathbf{x}_1 + b\mathbf{x}_2 + c\mathbf{x}_3 + d\mathbf{x}_4. \tag{12.60}$$

We construct matrices by leaving out two indices, and denoting by subscript the indices left out:

$$m_{12} = |\mathbf{x}_3\ \mathbf{x}_4\ \mathbf{x}_5|. \tag{12.61}$$

At this point, we simplify the notation,[8] get rid of the $\mathbf{x}$'s and just keep track of the subscripts, rewriting the definition of m_{12}:

$$m_{12} = |3\ 4\ 5|. \tag{12.62}$$

As above, we can use algebra to relate the determinants and the coefficients, for example,

$$\begin{aligned} m_{12} &= a|3\ 4\ 1| + b|3\ 4\ 2| && (12.63)\\ &= a|1\ 3\ 4| + b|2\ 3\ 4| && (12.64)\\ &= am_{25} + bm_{15} && (12.65) \end{aligned}$$

and

$$\begin{aligned} m_{13} &= am_{35} - cm_{15} && (12.66)\\ m_{14} &= am_{45} + dm_{15}. && (12.67) \end{aligned}$$

Eq. 12.59 presents forms for the coefficients in terms of the M_i's, and adding those relations into Eqs. 12.65–12.67 produces

$$\begin{aligned} M_5m_{12} + M_1m_{25} - M_2m_{15} &= 0\\ M_5m_{13} + M_1m_{35} - M_3m_{15} &= 0\\ M_5m_{14} + M_1m_{45} - M_4m_{15} &= 0. \end{aligned} \tag{12.68}$$

These relations are invariant to both 3D and 2D motions except for a multiplicative scale that affects all the M_i's the same. We can eliminate this dependence by using ratios and define 3D invariants

$$I_1 = \frac{M_1}{M_5} \quad I_2 = \frac{M_2}{M_5} \quad I_3 = \frac{M_3}{M_5} \tag{12.69}$$

and 2D invariants

$$i_{12} = \frac{m_{12}}{m_{15}} \quad i_{13} = \frac{m_{13}}{m_{15}} \quad i_{25} = \frac{m_{25}}{m_{15}} \quad i_{35} = \frac{m_{35}}{m_{15}}. \tag{12.70}$$

The denominators cannot be zero, since they are the determinants of matrices that we know are nonsingular. Look at Eq. 12.68 and divide the top line by M_5:

$$m_{12} + \frac{M_1}{M_5}m_{25} - \frac{M_2}{M_5}m_{15} = 0 \tag{12.71}$$

[8] Watch out for the notation change here!

which simplifies to

$$m_{12} + I_1 m_{25} - I_2 m_{15} = 0. \tag{12.72}$$

Similarly divide by m_{15} and produce two independent equations:

$$\begin{aligned} i_{12} + I_1 i_{25} - I_2 &= 0 \\ i_{13} + I_1 i_{35} - I_3 &= 0. \end{aligned} \tag{12.73}$$

If we have the 2D invariants we have two equations for the 3D invariants. The two equations of Eq. 12.73 do not, unfortunately, completely determine the three 3D invariants. Still those two equations determine a space line in the 3-space of the I's.

How do we use an idea like this? Given a 3D model of an object, and any five points, four of which are not coplanar, we can find I_1, I_2, and I_3, a set of three scalars, which define a point in a 3D space that we will call the "model space." To perform recognition, we first extract from the 2D image, (generally several) 5-tuples of feature points and from them construct the 2D invariants. Each 5-tuple gives rise to two equations in I_1, I_2, I_3 (Eq. 12.73) space, that is, a straight line in the 3D invariant space. If a 5-tuple in the 2D image is a projection of some 5-tuple in 3D, then the line so obtained will pass through the single point representing the model. If we have a different projection of those five points, we get a different straight line, but it still passes through the model point.

Implementing this for real-world scenes is slightly more complicated than this description because one must actually make use of projective geometry rather than assuming orthogonal projections. Other complications arise in determining a suitable way to choose 5-tuples, and a means for dealing with the fact that the line may "almost" pass through the point. Weiss and Ray [12.56] address these issues. This approach, we might point out, also provides a solution to the correspondence problem.

12.5 Controlling the Lighting – Range from One Camera and a Light

There are two principal ways to acquire a dense range image by controlling the lighting: using time of flight, and using structured illumination.

Time of flight If a laser or other light source can be modulated quickly, and if fast electronics can measure the time the light pulse took to get from the source laser, bounce off the distant target, and return to the detector, then the distance to the remote object can easily be calculated. This method works extremely well to create range images of distant objects like the moon. However, the speed of light is relatively rapid, and we don't have electronics fast enough to do this with small objects such as components on a circuit board. For very short distances, acquiring a range image will require structured lighting.

Design of time of flight systems requires skill in instrument design, which is beyond the scope of this book.

Structured Illumination A modification to stereo eliminates the correspondence problem: replace one camera with a light source (e.g., a laser beam passing through a cylindrical mirror creates a plane of light.). To see how this works, think of the

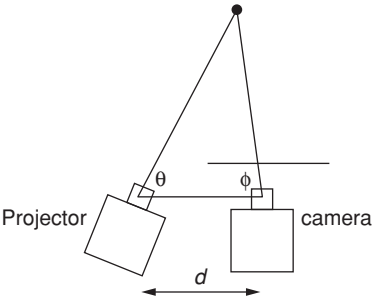


Figure 12.13 Structured Illumination

regular, two-camera stereopsis problem and replace one of those cameras by a projector that shines a very narrow, very bright slit on the scene, as illustrated in Figure 12.13.

Now, one angle, θ, is known from the projector; the other angle, ϕ, is measured by finding the bright spot in the camera image, counting over pixels, and knowing the relationship between pixels and angle. Finally, knowledge of the distance between cameras, d, makes the triangle solvable. An interesting problem occurs when one uses structured illumination to look at specular reflectors such as metal surfaces. With specular reflectors, either not enough or too much light may be reflected (polarization filters help [12.35]).

12.5.1 Structured Illumination – An Example

The key to doing structured illumination is to realize that by controlling the lighting, one or more of the unknowns in the stereopsis problem may be eliminated. Let's look at an example in more detail to see how this might work.

The problem to be solved is an application in robot vision: a robot is to pick up shiny, metallic turbine blades from a cart and place them into a machine for further processing. In order to locate the blades, a horizontal slit of light is projected onto the scene, by passing a laser beam through a cylindrical lens. The geometry of the resulting images is illustrated in Figure 12.14. If there were no blade in the image, the light stripe from the laser would form

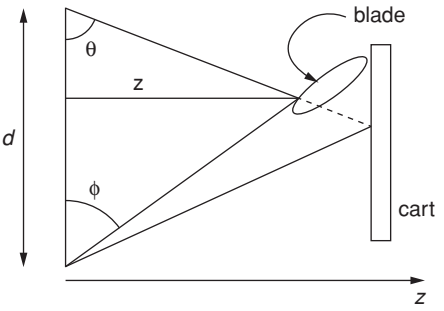


Figure 12.14 In an application requiring structured illumination to locate turbine blades, the presence of a blade translates the location of the light stripe vertically in the image. (From [12.35])

a horizontal line in the image as it is reflected off the cart. The presence of the blade causes a vertical translation of the light stripe. The amount of vertical displacement is directly proportional to an angular difference that produces the angle ϕ. Knowing the two angles and the distance d between the camera and the projector allows a simple calculation of the distance z.

$$z = \frac{d \tan\theta \tan\phi}{\tan\theta + \tan\phi}. \tag{12.74}$$

Although this relationship is relatively simple, it turns out to be even simpler to just keep a lookup table of z vs. row displacement.

One practical problem that arises in this problem is the specular nature of the reflections from specular reflectors such as turbine blades; the bright spots may be orders of magnitude brighter than the rest of the image. This is dealt with by passing the beam through a polarizing filter. By placing another such filter on the camera lens, the specular spots are considerably reduced in magnitude.

In this example of the use of structured illumination, only one light stripe was projected at a time, so there was no ambiguity about which bright spot resulted from which projector. However, in the more general case, one may have multiple light sources, and some method for disambiguation is required [12.7, 12.31].

12.6 Shape from x – Range from a Single Camera

In this section, *shape* is defined as the depth function $z(x, y)$. However it is not obvious how to convert measurements of brightness $f(x, y)$ to depth without special hardware. To accomplish that, a number of methods have been developed and those are described in section 12.6.1.

12.6.1 Shape from Shading

Shape from Shading was first introduced by Horn [12.18], who argued that some knowledge of how light is generated, reflected, and observed could substantially improve the performance of Computer Vision systems. Consider Figure 12.15, and assume you know

- a law governing how light is scattered
- the albedo (how the surface is painted/colored) of the surface
- the direction to the light source
- the direction to the observer
- the measured brightness of the pixel

Then the first objective is to obtain the surface normal. Once surface normals are known, it will become possible to actually determine the surface, $z(x, y)$.

The surface normal vector may be written as $\mathbf{n} = \frac{\mathbf{r}}{|\mathbf{r}|}$, where the direction vector $\mathbf{r} = \left[\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}, 1\right]^{\mathrm{T}}$. In most shape-from-shading literature, the partial derivatives are abbreviated using: $p = \frac{\partial z}{\partial x}, q = \frac{\partial z}{\partial y}$.

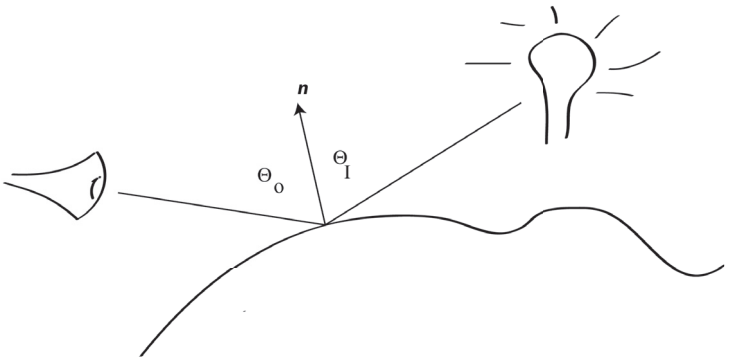


Figure 12.15 Light strikes a surface at an incident angle, relative to the surface normal, N, and is reflected/scattered in another direction.

Despite the fact that we use the term "brightness" constantly, it actually does not have a rigorous physical definition. Horn [12.18] reminds us that we should define *irradiance* as the power per unit area falling on a surface, measured in watts per square meter. Then, radiance is defined as the power per unit foreshortened area per unit solid angle. This dependence on foreshortened area makes it clear that both the angle of observation and the angle of incidence may play important roles in scene "brightness."

At a point, the *reflectivity model* of a surface relates the surface normal, the direction to the light source, and the direction to the observer. For example, the reflected brightness might be related to the incident brightness with a relationship such as

$$R(x, y) = r_0 f(x, y) \cos(\theta_I). \tag{12.75}$$

Thus, if we know the incident brightness f, the albedo (how the surface is painted) r_0, and the reflected brightness R, we might hope to solve for θ_I, and from that, infer the surface normal, and from that the surface. The reflectivity function of Eq. 12.75 is known as a *Lambertian* model. Note that the angle of observation does not enter the Lambertian model. Try this: look at a white styrofoam cup and you will see that no matter how you turn it in front of your eyes, the brightness does not change, so the apparent reflected light is independent of the angle of observation. However, put it close to a single light source and you will see that the angle of incidence does matter in how bright the surface appears.

Another familiar reflectivity function is the specular model

$$R(x, y) = r_0 f(x, y) \delta(\theta_I - \theta_o), \tag{12.76}$$

which describes mirrors – you only get a reflection if the angle of illumination equals the angle of incidence. Of course most surfaces, even "shiny" ones, are not perfect specular reflectors, and a more realistic model for a mixed surface might be

$$R(x, y) = r_0 f(x, y) \cos^4(\theta_I - \theta_o). \tag{12.77}$$

Although the use of reflectivity functions requires radiometric calibration of cameras [12.16], that requirement in itself is not the major difficulty. To see the complexity of the problem [12.33] let us expand Eq. 12.75 in terms of the elements of the unit incidence vector $\mathbf{I} = [I_x\ I_y\ I_z]^{\mathrm{T}}$ and the normal vector, $\mathbf{n}$, recalling that the inner product of

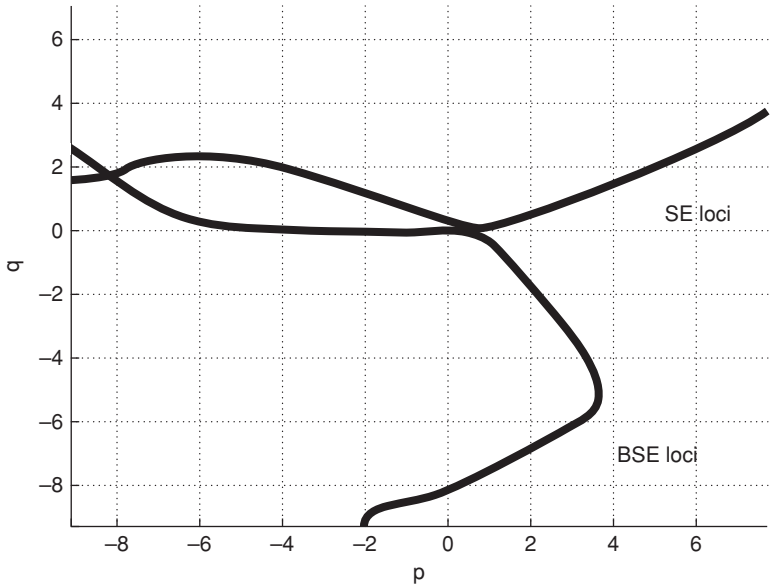


Figure 12.16 A given measured brightness pair can be created by a locus of p, q values in both SE and BSE images. (The authors are grateful to B. Karaçali for this figure.)

these two vectors will be $\cos\theta_I$.

$$R(x,y) = r_0 f(x,y)\mathbf{I}\cdot\mathbf{n} = r_0 f(x,y)\left(I_x\frac{\partial z}{\partial x} + I_y\frac{\partial z}{\partial y} + I_z n_z\right). \tag{12.78}$$

Assuming we know the angle of incidence θ (which we will actually know only approximately), the albedo r_0, and the incident brightness f, we still have a partial differential equation that we must solve to determine the surface function z.

So how do we do that – solve this ugly partial differential equation? We can do it if we have p and q, but that is a challenging problem in itself. There are a variety of approaches, (See the text by Horn [12.18]) the details of which are beyond the scope of this book. Basically, a numeric differential equation solver is used to estimate $z(x,y)$ taking into account the boundary conditions of the object being estimated, as well as agreeing with the local gradient information provided by shading.

12.6.2 Using Shape from Shading with Two Light Sources

Here we illustrate how to use shape-from-shading when two light sources are available. We will not go into the physics here, but scanning electron microscopes provide an excellent example application. In such a microscope two images are produced, one from secondary emission (SE) electrons and one from backscattered (BSE) electrons.

Rather than attempting to model the geometry of the camera(s) exactly, we may simply measure the reflectivity functions RSE and RBSE by placing a sphere in the microscope and imaging it. On a sphere, p and q are easy to determine. For example, a particular brightness value might be represented by the locus of points shown in Figure 12.16. Then,

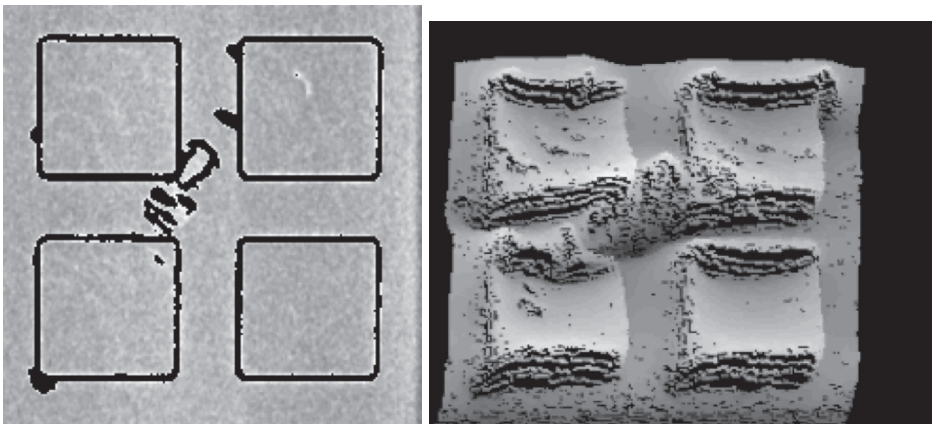


Figure 12.17 An integrated circuit with 4 pads is illustrated on the left from the backscattered electron sensor. Combining with the secondary emission sensor and using the techniques of this section allows a range image to be produced, which is illustrated on the right. With this information, the heights of the pads and the defect can be easily determined. (The authors are grateful to B. Karaçali for this figure.)

assuming the albedo is known (that is an important assumption), the measured brightness is a function of p and q in both the BSE and SE images. Although there is an infinite number of possible p, q values in each image, there are only two possible (p, q) pairs that can explain both the measured brightnesses. We thus solve for $z(x, y)$ by defining an objective function and finding the surface that minimizes that function.

To accomplish this, first let $\rho((p_1, q_1), (p_2, q_2))$ define a function that represents the difference in the surface normals defined by the two p, q ordered pairs. (it could, for example, be something as simple as the cosine of the angle between them. Then, we consider the difference between the surface normal defined by the i^{th} (p, q) pair and the vector that is normal to the surface $z(x, y)$, and write that difference as $d_i(x, y) = \rho\left(\left(\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}\right), (p, q)\right)$. Finally, assuming that the two curves of Figure 12.16 intersect at m points (and making m a function of x, y to remind the reader that all this is being done for a single x, y point), we define an objective function as

$$E = \sum_{x,y} \left(\sum_{i}^{m(x,y)} (d_i(x, y))^{-1} \right)^{-1} + \lambda \Lambda \tag{12.79}$$

allowing Λ to be a regularization term such as piecewise linearity.

Use of this technique to inspect integrated circuits[12.21] is illustrated in Figure 12.17.

A variation on the simple model given above occurs when the surface is not totally reflective. For example, at infrared wavelengths (and other wavelengths, to a lesser extent), the power measured coming from a surface is a combination of energy that is reflected and energy that is emitted (so called "black body radiation") as a result of the temperature of the surface[12.29].

12.6.3 Shape from Surface Normals

Suppose we have the two partial derivatives, $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$; we have the derivative of z and we want z – some calculus professor somewhere said we should integrate. Let's try to do that. Since the objective of this book is to illustrate simple principles and not go into sophisticated math any more than necessary, we present a simple strategy here. No, it's not the best, but it should illustrate the principles.

1. Start with $x = 0$ and $y = 0$
2. Set pixel (x, y) to zero.
3. Range over values of y
   (a) range over values of x
      - set $z(x, y) \leftarrow z(x-1, y) + p(x-1, y)$.
   (b) increment y and set x to zero.
   (c) set $z(x, y) \leftarrow z(x, y-1) + q(x, y-1)$

Observe that the algorithm above is going through the image in a raster scanning way, incrementing z by p at every point first, and then doing one increment by q per line. There are other ways to walk through the image, other *paths* to take. If it should turn out that we get the same image z no matter which path we take, we say that this is an *integrable* function. In order to be integrable, a function should be continuous and differentiable.

Many papers and Horn's classic text [12.18] address approaches for solving various special cases of the shape from shading problem. A paper by Zhang [12.61] surveys the field between Horn's book and 1999. (Remember, Eq. 12.78 is itself a special case – it assumes the brightness does not depend on the observation angle). In the following, we discuss another special case, photometric stereo.

12.6.4 Photometric Stereo

In many cases, it is reasonable to model the reflectivity of a surface as proportional to the cosine of the angle between the surface normal vector and the illumination vector, a Lambertian reflectance model:

$$R(x, y) = r_o \mathbf{N}_i \cdot \mathbf{n} \tag{12.80}$$

where $\mathbf{N}_i$ is a unit vector[9] in the direction of light source i, and $\mathbf{n}$ is the unit normal to the surface. If we are fortunate enough to have an object, a Lambertian reflector, which satisfies this equation, and which possesses the same albedo (r_0), independent of the illumination, we can make use of multiple pictures from multiple angles (not all sources in a single straight line) to determine the surface normal [12.19, 12.57]. Let us illuminate a particular pixel with three different light sources (one at a time) and measure the brightness of that pixel each time. At a given pixel, we construct a vector from the three observations

$$\mathbf{R} = [R_1,\ R_2,\ R_3]^{\mathrm{T}}. \tag{12.81}$$

[9] Huh? Earlier we said "cosine." There's no cosine in Eq. 12.80. How can this be explained?

We know the direction of each light source.[10] Let those directions be denoted by unit vectors from the surface point toward the light source, $\mathbf{N}_1$, $\mathbf{N}_2$, and $\mathbf{N}_3$. Write those three direction vectors in a single matrix by making each vector a row of the matrix.

$$N = \begin{bmatrix} \mathbf{N}_1 \\ \mathbf{N}_2 \\ \mathbf{N}_3 \end{bmatrix} = \begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{bmatrix}. \tag{12.82}$$

and now we have a matrix version of Eq. 12.80.

$$\mathbf{R} = r_0 N \mathbf{n}. \tag{12.83}$$

We can find the scalar albedo, r_0 by multiplying both sides of Eq. 12.83 by N^{-1} to get

$$N^{-1}\mathbf{R} = r_0 \mathbf{n}. \tag{12.84}$$

N is known, and $\mathbf{n}$ is a unit vector, so we take absolute values:

$$r_0 = |N^{-1}R|, \tag{12.85}$$

and once we know r_0, we can solve for $\mathbf{n}$ using

$$\mathbf{n} = \frac{1}{r_0} N^{-1}\mathbf{R}. \tag{12.86}$$

This derivation of photometric stereo assumes the albedo (sometimes called the surface reflectance) is the same for every angle of illumination. In the next subsection, we illustrate an application that combines shape from shading with photometric stereo, and does not make this assumption. For example, specular reflectors provide a special condition: the angle of observation is exactly equal to the angle of incidence. This allows special techniques to be used [12.34]. However, first we consider what happens if more than three light sources are available.

12.6.5 Photometric Stereo from More Than Three Sources

What if we used more than three light sources? This gives us a wonderful context in which to discuss an important topic, overdetermined systems and the *pseudoinverse*. If we in fact use more than three light sources, we would hope to cancel out some effects of noise and/or measurement error. Suppose we have k light sources. Then Eq. 12.83 is rewritten as $\mathbf{R}_{k\times 1} = N_{k\times 3}\mathbf{n}_{3\times 1}$ where subscripts are used to emphasize the matrix dimensions, and r_0 is removed for clarity of explanation.

Now, we cannot simply multiply by the inverse of N because N is not square. It takes only three equations to determine the surface normal. With more than three equations, our problem is *overdetermined*. As we have done so many times before, let us set up a minimization problem: find the surface normal vector $\mathbf{n}$ that minimizes the squared sum of differences between measurements R and products of N and $\mathbf{n}$.

Of course, if Eq. 12.80 is strictly true everywhere, then we do not need to do a minimization. Of course, if Eq. 12.80 were true everywhere, there would be no point to taking

[10] Well, almost. Since the pixels are not all at the same point, there is some variation in direction from pixel to pixel, but the light source is so far away (normally) that the differences in direction are negligible.

more than three measurements. We believe that measurements are not perfect and there is some advantage to taking additional ones. First, define a vector $\mathbf{R} = [R_1 R_2 \ldots R_k]^{\mathrm{T}}$. Define an objective function E that incorporates the desire to find an optimal solution:

$$E = \sum_{i=1}^{k} (R_i - \mathbf{N}_i \mathbf{n})^2. \tag{12.87}$$

Using each of the $\mathbf{R}_i$ as a column, construct a matrix denoted R. Then we have

$$E = (R - N\mathbf{n})^{\mathrm{T}}(R - N\mathbf{n}). \tag{12.88}$$

Expand the product

$$E = R^{\mathrm{T}}R - 2\mathbf{n}^{\mathrm{T}}N^{\mathrm{T}}R + \mathbf{n}^{\mathrm{T}}N^{\mathrm{T}}N\mathbf{n}. \tag{12.89}$$

We wish to find the surface normal vector $\mathbf{n}$ that minimizes this sum-squared difference E, so we differentiate E with respect to $\mathbf{n}^{\mathrm{T}}$,

$$\nabla_{\mathbf{n}} E = -2N^{\mathrm{T}}R + 2N^{\mathrm{T}}N\mathbf{n} \tag{12.90}$$

and setting the gradient to zero we have

$$N^{\mathrm{T}}N\mathbf{n} = N^{\mathrm{T}}R \tag{12.91}$$

or

$$\mathbf{n} = (N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}}R. \tag{12.92}$$

(In case you have not recognized it yet, the pseudoinverse just appeared). That was a lot of work. Let's see if there is an easier way:

Go back to Eq. 12.83, again omitting the r_0 for clarity, and multiply both sides by N^{T}.

$$N^{\mathrm{T}}R = N^{\mathrm{T}}N\mathbf{n}. \tag{12.93}$$

Multiply both sides by $(N^{\mathrm{T}}N)^{-1}$ and we find the same result as Eq. 12.92.

So why did we go to so much trouble – all the work of Eqs. 12.87 through 12.91 seems like a waste. We have now demonstrated to you that multiplying by the pseudoinverse $(N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}}$ produces the minimum squared error estimate of an overdetermined linear system. That is a significant result, important in the case of photometric stereo, and in many other applications as well.

12.6.6 Shape from Texture

Texture, or rather spatial variations in texture, can be used to characterize 3D shape, as illustrated in Figure 12.18. For a very nice collection of pictures showing how texture suggests 3D shape, see [12.49] Of course, in order to perform shape from texture, one must be able to robustly extract texture primitives. This topic, however, is too broad for the scope of this book.

The ideas of shape from texture have also found application in work to recover 3D motion [12.46, 12.44].

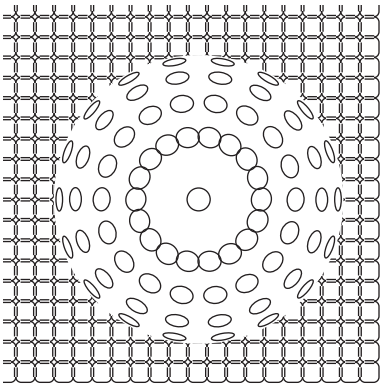


Figure 12.18 This figure contains no shading or color, only variations in texture, yet still presents the viewer with the perception of a three dimensional ball on a flat plane. (The authors are grateful to John Franke for his help in developing the software to generate this figure.)

12.6.7 Shape from Focus

That one can obtain range from focus is obvious, however, the extension to robust shape from focus is difficult. The principal problem is determining precisely when each pixel is in focus [12.32, 12.45]. Sadly, we are also not able to cover shape from focus in this book.

12.7 Surfaces in 3-Space

We have spent a substantial part of in this chapter discussing how to obtain range information about a surface. But once we have it, what do we do with it?

In this section, surfaces are discussed. A surface (as it is defined here) is a two-parameter manifold embedded in 3-space. We cannot constrain our surfaces to be differentiable, because true discontinuities can occur. Nonetheless, we think of the surfaces with which we deal to be at least locally differentiable. We begin with second-order surfaces.

12.7.1 Quadric Surfaces

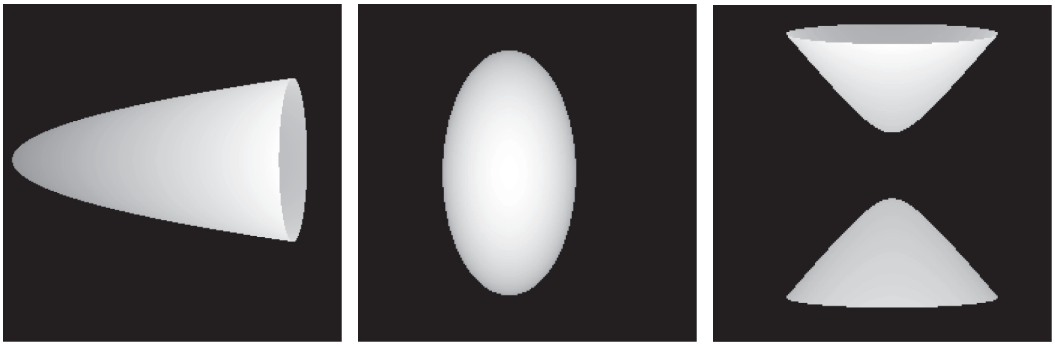
A surface defined by an algebraic equation of degree two is called a *quadric*. The general equation for a quadric is

$$ax^2 + by^2 + cz^2 + fyz + gzx + hxy + px + qy + rz + d = 0. \tag{12.94}$$

This one equation describes all the second order surfaces.

Quadrics: Three Examples

Three example surfaces are illustrated in Figure 12.19. The range image simulation software uses the quadric coefficient matrix of Eq. 12.104, but specifies the right column



(a) Synthetic range image of a paraboloid, terminated by a plane. (b) Synthetic range image of an ellipsoid (c) Synthetic range image of a hyperboloid of two sheets.

Figure 12.19 Example quadric functions. Coefficient matrices are given in the text.

(translations) in a different way. Values for the matrix are

$$\textbf{Paraboloid} \begin{bmatrix} 0 & .05 & .2 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \quad \text{(Fig. 12.19a)}$$

$$\textbf{Ellipsoid} \begin{bmatrix} .51 & .15 & .2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{(Fig. 12.19b)}$$

$$\textbf{Hyperboloid} \begin{bmatrix} 1.5 & -1 & 5 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \quad \text{(Fig. 12.19c)}$$

If the quadric is centered at the origin, and its principal axes happen to be aligned with the coordinate axes, the quadric will take on a particular form. For example, an ellipsoid aligned with the coordinate axes has the special form

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1. \tag{12.95}$$

The a, b, and c of the previous equation are simply scalar constants, and not the a, b, and c of Eq. 12.94.

However, when the axes of the quadric do not align with the coordinate axes, only the general form of Eq. 12.94 occurs.

12.7.2 Fitting Quadric Surfaces to Data

In range images, we typically have numerous surfaces. One may simply seek surfaces that do not bend too quickly. Following that philosophy produces algorithms that seek smooth solutions and segment regions along lines of high surface curvature. One example of this philosophy was discussed in section 6.5.2 where we describe an algorithm that removes noise while seeking the best piecewise linear fit to the data points. Such a fit is equivalent to fitting a surface with a set of planes. Points where planes meet produce either "roof" edges or "step" edges, depending on viewpoint (see Figure 5.9). If an annealing algorithm

is used such as MFA, good segmentations to more general surfaces can be produced simply by not running the algorithm all the way to a truly planar solution [12.3].

A second philosophical approach to range image segmentation is to assume some equation for the surface, e.g., a quadric (a general second order surface, defined in section 12.7.1). Then, all points that satisfy that equation and that are adjacent belong to the same surface. This philosophy mixes the problems of segmentation and fitting [12.6, 12.39].

Below, we choose a form to use to describe a surface. We have two choices for the type of description, implicit and explicit. The implicit form is clearly much more attractive, but can be much more difficult to fit. For example, consider second order forms. An explicit representation for a surface might be

$$z = ax^2 + by^2 + cxy + dx + ey + g, \tag{12.96}$$

and an implicit form is

$$S(x, y, z) = ax^2 + by^2 + cz^2 + fyz + gzx + hxy + px + qy + rz + d = 0. \tag{12.97}$$

As mentioned in section 12.7.1, the expression of Eq. 12.97 is called a quadric, and is a general form that describes all second order surfaces (cones, spheres, planes, ellipsoids, etc.). In Chapter 5 you learned how to fit an explicit function to data, by minimizing the squared error. Unfortunately, the explicit form does not easily allow higher order terms in z. You could solve Eq. 12.97 for z, using the quadratic equation, and then have an explicit form, but now you have a square root on the right-hand side, and lose the ability to use linear methods for solving for the vector of coefficients.

However, the implicit form can be used making the following observation: If the point $[x_i, y_i, z_i]^{\mathrm{T}}$ is on the surface described by the parameter vector $[a, b, c, f, g, h, p, q, r, d]^{\mathrm{T}}$, then $S(x_i, y_i, z_i)$ should be exactly zero.

When we substitute an actual x, y, z value into the equation for a particular quadric it is likely that the value of S will not be precisely zero. The value of S is known as the *residual*. We use the fact that the residual should be zero if the data and the coefficients agree in order to develop an algorithm for finding the coefficients, which is sometimes known as *minimizing the squared residual*, and sometimes as *minimizing the algebraic distance*.

The coefficients may be found by minimizing $E = \sum_i (S(x_i, y_i, z_i))^2$. In some cases, this gives good results, but it is not really what we want. We really should be minimizing some distance metric, for example the Euclidean distance from the point to the surface (this is known as the *geometric distance* [12.47] to the surface). This often turns out to be algebraically intractable. (To implement this, see [12.51] for important details.) Although methods based on the algebraic distance work relatively well most of the time, they can certainly fail. Any distance measure we use should have the following properties [12.27]: (1) the measure should be zero whenever the true (Euclidean, geometric) distance is zero (the algebraic distance accomplishes this); (2) at the sample points, the derivatives with respect to the parameters are the same for the true distance and the measure.

In the next subsection, the quadric is fit to a set of data.

Fitting the Quadric

We begin with a version of the quadric equation that has been normalized by dividing by the constant term:

$$ax^2 + by^2 + cz^2 + fyz + gzx + hxy + px + qy + rz + 1 = 0. \tag{12.98}$$

For some particular point (x_i, y_i, z_i) define the vectors:

$$\mathbf{x}_i = \begin{bmatrix} x_i^2 \\ y_i^2 \\ z_i^2 \\ y_i z_i \\ z_i x_i \\ x_i y_i \\ x_i \\ y_i \\ z_i \end{bmatrix} \quad \text{and} \quad \mathbf{a} = \begin{bmatrix} a \\ b \\ c \\ f \\ g \\ h \\ p \\ q \\ r \end{bmatrix}.$$

Construct a matrix X from the set of points $\mathbf{x}_i$ by making each $\mathbf{x}_i$ a column of X. Now, X is a matrix of 9 rows and n columns, where n is the number of points we measure. If each point (each column of X) actually lies on the surface defined by $\mathbf{a}$, then

$$X^{\mathrm{T}}\mathbf{a} + [\mathbf{1}] = [\mathbf{0}] \tag{12.99}$$

where $[\mathbf{1}]$ and $[\mathbf{0}]$ are n-vectors of ones and zeros, respectively.

or

$$X^{\mathrm{T}}\mathbf{a} = [-\mathbf{1}]. \tag{12.100}$$

Now, we may use the pseudoinverse to find the mean-squared estimate of $\mathbf{a}$

$$\hat{\mathbf{a}} = (XX^{\mathrm{T}})^{-1}X[-\mathbf{1}] \tag{12.101}$$

Rewriting Eq. 12.101 to show dimensions, we see

$$\hat{\mathbf{a}}_{9\times 1} = \left(X_{9\times n}X^{\mathrm{T}}_{n\times 9}\right)^{-1}_{9\times 9} X_{9\times n}[-\mathbf{1}]_{n\times 1} \tag{12.102}$$

which requires only the solution of a 9×9 inverse to find the coefficients that describe this quadric. Furthermore this method allows use of an overdetermined system of equations and a minimum squared error solution.

Determining the Geometry

From range or other surface data, the quadric coefficients may be determined by methods such as those in the previous section. Given the coefficients, the type of quadric may be determined by the following method. (Note that here we define the yz, zx, and xy terms using an additional coefficient of 2. This is just to make the matrix of Eq. 12.104 work out conveniently. Otherwise, the 2's will not be there.[11])

[11] The authors are grateful to the late Dr. G. L. Bilbro for his formulation of this method.

1. If there is a constant term, d, divide by the constant, redefining the other coefficients (e.g., $a = \frac{a}{d}$), resulting in a form of the quadric equation in which the constant term is unity:

$$ax^2 + by^2 + cz^2 + 2fyz + 2gzx + 2hxy + px + qy + rz + 1 = 0. \tag{12.103}$$

   This equation can then be written in the following form

$$[x\ y\ z\ 1] \begin{bmatrix} a & h & g & p \\ h & b & f & q \\ g & f & c & r \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0. \tag{12.104}$$

2. Consider the upper left 3×3 submatrix. Obtain its three eigenvalues, λ_1, λ_2, and λ_3. Then find the reciprocal of each that is nonzero, $r_1 = \frac{1}{\lambda_1}$, $r_2 = \frac{1}{\lambda_2}$, and $r_3 = \frac{1}{\lambda_3}$.
3. At least one reciprocal must be positive to have a real surface.
4. If exactly one is positive then the surface is a hyperboloid of two sheets.
5. If exactly two are positive, then it is a hyperboloid of one sheet.
6. If all three are positive, it is an ellipsoid, and the square roots of r_1, r_2, r_3 are the major axes of the ellipsoid.
7. Otherwise the distance between the foci of hyperboloids is determined by the magnitudes of the r's.

12.7.3 Fitting Ellipses and Ellipsoids

Although this section discusses fitting surfaces, we have introduced the concept of algebraic distance, and this is an appropriate place to consider again the simpler case of fitting ellipses to curve data, as well as the three-dimensional extension to fitting ellipsoids. An ellipse is described by the general equation for a conic section:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0. \tag{12.105}$$

This implicit form describes not only ellipses, but lines, hyperbolae, parabolas, and circles. To guarantee that the resulting curve is an ellipse we must also ensure that it satisfies

$$b^2 - 4ac < 0. \tag{12.106}$$

Satisfying this constraint produces an optimization problem that is nonlinear. Instead, we could use residuals to find the coefficients $a - f$ exactly as was done in the previous section.

Following this approach provides a solution that works reasonably well but tends to fit areas of low curvature with hyperbolic arcs rather than with ellipses. Similar difficulties occur when attempting to fit ellipsoids to range data. See Wang [12.54], Rosin and West [12.36], and Fitzgibbon et al. [12.9] for more details.

There is a fast and easy solution to finding ellipses, which works well if neither end of the major axis is occluded, which we already discussed in Section 9.4, and another method by Lei and Wong [12.58], which uses accumulators in a powerful way.

12.8 Conclusion

Many papers have been written that extract three-dimensional shape from various sources: from silhouettes [12.23, 12.24, 12.26, 12.30, 12.62]; from images of specular reflectors [12.37]; from three orthogonal projections (x-ray projections) [12.48], making use of the assumption that objects tend to have orthogonality [12.12], or symmetry [12.10]. Ultimately, in all these algorithms one must address the question of visibility [12.50]. That is, in any particular image, not every part of every surface is visible to the camera, since occlusion may occur.

How you fit a function to data also depends on the nature of the noise or corruption to the data. If the noise is additive, zero-mean Gaussian (which is what we almost always assume) then the minimum vertical distance (Minimizing the Sum-Squared Error) or the minimal normal distance (which we called eigenvector line fitting) methods work well.

If the noise is not Gaussian, then other methods are more appropriate. For example, nuclear medicine images are corrupted primarily by counting (Poisson) noise. Such a noise differs from Gaussian in two important ways: it is never negative, and it is signal dependent. Well away from zero, Poisson noise may be reasonably modeled by additive Gaussian with a variance equal to the signal. Other sensors produce other types of noise. Stewart [12.43] considers the case of inliers and outliers, but assumes that the bad data are randomly distributed over the dynamic range of the sensor. That is, the noise is not additive.

Given one segmentation, should you merge two adjacent regions? If they are adjacent and satisfy the same equation to within some noise measure, they should be merged. Many flavors of this philosophy exist: [12.5, 12.22, 12.25, 12.38]. Other relevant papers on fitting surfaces include [12.2, 12.55, 12.59].

One is also faced with the issue of what surface measurements to use as a basis for segmentation. Curvature would appear to be particularly attractive since the measurement curvature is invariant to viewpoint. However, "curvature estimates" are very sensitive to quantization noise [12.53] since they are based on second derivatives.

Generalized Cylinders (GC)

A cylinder may be described as a circle that translates along a straight line in space, with the plane of the circle perpendicular to the line. Now, suppose that the line is allowed to bend in space, in fact to become an arbitrary space curve, parameterized by arc length, s. Then, the line becomes a vector function of s, $[x(s), y(s), z(s)]$. Next, allow the radius of the circle to vary with the point along the curve, $R = R(s)$, and you have some idea of what a generalized cylinder is [12.4, 12.11, 12.13, 12.40, 12.41, 12.60]. However, the concepts of GC's are more general even than described above. The object translated does not have to be a circle, it can be any 2D shape.

If we can fit GC's to regions, we can then use the vector function of the line and the radius function as features to describe the shape of the region. However, there are significant challenges to fitting GC's to images. We will not pursue the GC idea any further in this book.

Self-occlusion

Range images, one might think, already contain a complete description of three-dimensional shape, but, of course, you cannot see the entire surface in one image [12.17]. A tough problem is how to integrate several range images to form one description of a three-dimensional object [12.42].

Even though you may have segmented surfaces with some success, those segmentations are almost never perfectly correct. One might think that the equations describing the intersection of space curves would be straightforward to find. After all, you have the equations of the surfaces whose intersections determine the edges. Just calculate the intersection![12] Alas, it is never quite that easy. The problems occur when you have vertices, the intersections of edges: the trihedral or multihedral intersection points. Those equations you just derived never intersect precisely at a point. Hoover et al. [12.17] addresses this problem, and extends the solution to nonvisible surfaces.

12.9 Assignments

Assignment 12.1: A range camera is mounted at the origin, facing down the positive Z axis. This is a scanning time-of-flight laser scanner. The laser is mounted on a platform that can rotate about two different axes. First, it can can rotate about the Z axis through an angle ϕ. Then it can rotate about $\hat{y}$ axis through an angle θ. The $\hat{y}$ is the y axis after the first rotation occurred. The beam hits a surface, and a detector measures the time-of-flight, and from that determines the distance to the surface, ρ.

Given these two angular values and the distance measurement,

1. Where is that point in 3-space? Yes, this is the transformation from spherical to Cartesian coordinates. But, don't just look it up, derive it.
2. Is it possible that there may exist points in the 3-space where we cannot accurately determine the Cartesian value? If so, describe them.

Assignment 12.2: In this chapter, you were introduced to the following notation: $[\mathbf{a}]_\times$ that was claimed to be a 3×3 matrix. Evaluate the product $[\mathbf{a}]_\times \mathbf{b}$, where $\mathbf{b}$ is a 3×1 vector. Compare the result of this operation with the result of computing the cross product of $\mathbf{a}$ with $\mathbf{b}$.

Assignment 12.3: The camera in the previous problem is built using electronics that have a rise time of 250ps. That is, from the time a pulse of light hits the detector, it is 250 picoseconds before the electronics outputs a logic "1". The fall time is the same.

What is the *range resolution* of this camera? That is, what is the smallest difference in range that this camera can detect? You may need to make some assumptions.

Assignment 12.4: In section 12.2.2 a simple camera is described in terms of a collection of parameters. Two parameters that are not provided in that set are the extent of the focal plane in the x and y directions. To see the effect of these two parameters, suppose you are

[12] They may not intersect because they are in 3D space.

presented with a camera in the central projection that has a focal length of 10mm. Suppose that camera has a focal plane that ranges as $-6mm < x < 6mm$ and $-4mm < y < 4mm$.

In section 12.2.2 we assumed the focal plane was infinite, but now we have constrained the focal plane. Does this limit what this camera can see? For example, can this camera see an object at the point $\mathbf{X} = [700mm, 300mm, 1.0m]^{\mathrm{T}}$? Describe the region of space that is visible by this camera.

Assignment 12.5: Someone hands you a projective camera with a focal length of 25mm, asks you to place it at $X = [100, 100, 10]^{\mathrm{T}}$, pointing down the Z axis, and then asks "can this camera see the origin without being rotated? What is your response?

He then asks you to find a position where the camera can be located such that it can see the origin, without rotating it. What do you say?

Assignment 12.6: The camera in the previous problem points straight down the Z axis. What projection matrix describes this camera?

Assignment 12.7: The classmate on your left dropped his exam paper on the floor during the test. On that, you "notice" that he has derived the following M matrix:

$$\begin{bmatrix} .25 & 0 & .75 & 0 \\ 0 & 1 & 0 & 0 \\ .75 & 0 & .25 & 0 \end{bmatrix}.$$

What do you think?

Assignment 12.8: Suppose you successfully found the fundamental matrix, $F = \begin{bmatrix} .25 & 0 & .75 \\ 0 & 1 & 0 \\ .75 & 0 & .25 \end{bmatrix}$ describing a pair of cameras. What is the equation of the epipolar line in the left-hand image corresponding to point $[8,\ 10\ 12]^{\mathrm{T}}$ in the right-hand image.

Assignment 12.9: Suppose the matrix F is the fundamental matrix describing the geometry of two cameras viewing the same scene. $\mathbf{x}_l$ is the homogeneous coordinate vector of $x - y$ coordinates in camera l. Similarly, $\mathbf{x}_r$ is the $x - y$ coordinates in camera r. What do the following conditions mean? (one possible answer is "nothing").

a) $\mathbf{x}_l^{\mathrm{T}}\mathbf{x}_r = 0$ _____
b) $\mathbf{x}_l F \mathbf{x}_r = 0$ _____
c) $\mathbf{x}_l^{\mathrm{T}}\mathbf{x}_r = 1$ _____
d) $\mathbf{x}_l^{\mathrm{T}}\mathbf{x}_l = 1$ _____

Assignment 12.10: Here's an alternative approach to doing stereopsis: Consider the line from the origin of the left camera through the point $\mathbf{x}_l$, and on to the point $\mathbf{X}$. That is a line in space, and has the form $ax + by + cz = r$. Given the location of the origin of that camera and the point $\mathbf{x}_l$, I can probably find the parameters of that equation. Similarly, I can find the equation of the point $\mathbf{x}_r$, resulting from projecting $\mathbf{X}$ only the right camera. Those are two lines in space, both passing through $\mathbf{X}$. To find $\mathbf{X}$, all I need to do is find the intersection of those two lines.

Is this a good approach? Will it work? Why or why not?

Assignment 12.11: A camera with a focal length of 10 is located so that its projection matrix is

$$P_1 = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

How could this camera be moved so it points directly at the point $[30 \;\; 0 \;\; 30]^{\mathrm{T}}$? After such a motion, what would the projection matrix be? Select all that are correct.

a) $P_{new} = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

b) $P_{new} = \begin{bmatrix} 10 & 0 & 0 & -300 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

c) $P_{new} = \begin{bmatrix} 10\cos(45^{\mathrm{o}}) & 0 & -\sin(45^{\mathrm{o}}) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(45^{\mathrm{o}}) & 0 & 10\cos(45^{\mathrm{o}}) & 0 \end{bmatrix}$

d) $P_{new} = \begin{bmatrix} 10\cos(45^{\mathrm{o}}) & \sin(45^{\mathrm{o}}) & 0 & -300 \\ -\sin(45^{\mathrm{o}}) & 10\cos(45^{\mathrm{o}}) & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$

Assignment 12.12: In this chapter, we introduced the pseudoinverse that could be used to "solve" (that is, find a minimum-squared-error estimate of the solution to) a linear equation of the form

$$C = AB$$

where A is a non-square matrix, and B and C are vectors of different lengths. What if instead the matrix is actually square? One would like to believe that, in this case, the pseudoinverse is the same as the inverse. Is this actually true? If so, prove it.

Assignment 12.13: Using the philosophy of photometric stereo, what is the minimum number of images required if the user insists on using an overdetermined system?

Assignment 12.14: Discuss in detail how Eq. 12.84 becomes Eq. 12.85.

Assignment 12.15: The following equation describes a surface in 3-space:

$$1.5x^2 + 1.5y^2 + 3z^2 + xy + x + z + 1 = 0.$$

What type of surface is this (ellipsoid, cylinder, hyperboloid, etc.?)

Bibliography

[12.1] M. Armstrong, A. Zisserman, and R. Hartley. Self-calibration from image triplets. *Computer Vision – ECCV'96*, pages 1–16, 1996.

[12.2] J. Berkmann and T. Caelli. Computation of surface geometry and segmentation using covariance techniques. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(11), 1994.

[12.3] G. Bilbro and W. Snyder. Range image restoration using mean field annealing. In *Advances in Neural Network Information Processing Systems*. Morgan-Kaufmann, 1989.

[12.4] T. Binford. Visual perception by computer. In *IEEE Conf. on Systems and Control*, December 1971.

[12.5] G. Blais and M. Levine. Registering multiview range data to create 3d computer objects. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(8), 1995.

[12.6] K. Boyer, M. Mirza, and G. Ganguly. The robust sequential estimator: A general approach and its application to surface organization in range data. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(10), 1994.

[12.7] D. Caspi, N. Kiryati, and J. Shamir. Range imaging with adaptive color structured light. *IEEE Trans. Pattern Anal. and Machine Intel.*, 20(5), May 1998.

[12.8] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with application to image analysis and automated cartography. *Communications of the ACM*, 24(6), 1981.

[12.9] A. Fitzgibbon, M. Pilu, and R. Fisher. Direct least square fitting of ellipses. *IEEE Trans. Pattern Anal. and Machine Intel.*, 21(5), May 1999.

[12.10] P. Flynn. 3-d object recognition with symmetric models: Symmetry extraction and encoding. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(8), August 1994.

[12.11] R. Gonzalez and P. Wintz. *Digital Image Processing*. Pearson, 1977.

[12.12] A. Gross. Toward object-based heuristics. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(8), August 1994.

[12.13] A. Gross and T. Boult. Recovery of SHGCs from a single intensity view. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(2), February 1996.

[12.14] G. Gurevich. *Foundations of the Theory of Algebraic Invariants*. P. Nordcliff Ltd., The Netherlands, 1964.

[12.15] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision, second edition*, volume 2. Cambridge University Press, 2004.

[12.16] G. Healey and R. Kondepudy. Radiometric CCD camera calibration and noise estimation. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(3), March 1994.

[12.17] A. Hoover, D. Goldgof, and K. Bowyer. Extracting a valid boundary representation from a segmented range image. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(9), September 1995.

[12.18] B. K. P. Horn. *Robot Vision*. MIT Press, 1986.

[12.19] Y. Iwahori, R. Woodham, and A. Bagheri. Principal components analysis and neural network implementation of photometric stereo. In *IEEE Conf. on Physics-Based Modeling in Computer Vision*, 1995.

[12.20] B. Julesz. *Foundations of Cyclopean Perception*. University of Chicago Press, 1971.

[12.21] B. Karaçali and W. Snyder. Noise reduction in surface reconstruction from a given gradient field. *International Journal of Computer Vision*, 60(1), October 2004.

[12.22] K. Koster and M. Spann. MIR: An approach to robust clustering – application to range image segmentation. *IEEE Trans. Pattern Anal. and Machine Intel.*, 22(5), 2000.

[12.23] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(2), February 1994.

[12.24] A. Laurentini. How far 3d shapes can be understood from 2d silhouettes. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(2), February 1995.

[12.25] S. LaValle and S. Hutchinson. A Bayesian segmentation methodology for parametric image models. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(2), 1995.

[12.26] S. Lavallee and R. Szeliski. Recovering the position and orientation for free-form objects from image contours using 3d distance maps. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(4), April 1995.

[12.27] K. Lee, Y. Choy, and S. Cho. Geometric structure analysis of document images: A knowledge-based approach. *IEEE Trans. Pattern Anal. and Machine Intel.*, 22(11), 2000.

[12.28] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, September 1981.

[12.29] J. Michel, N. Nandhakumar, and V. Velten. Thermophysical algebraic invariants from infrared imagery for object recognition. *IEEE Trans. Pattern Anal. and Machine Intel.*, 19(1), January 1997.

[12.30] F. Mokhtarian. Silhouette-based isolated object recognition through curvature scale space. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(5), May 1995.

[12.31] R. Morano, C. Ozturk, R. Conn, S. Dubin, S. Zietz, and J. Nissano. Structured light using pseudorandom codes. *IEEE Trans. Pattern Anal. and Machine Intel.*, 20(3), March 1998.

[12.32] S. Nayar and R. Bolle. Reflectance based object recognition. *Int. J. of Computer Vision*, 1996.

[12.33] S. Nayar and Y. Nakagawa. Shape from focus. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(8), August 1994.

[12.34] M. Oren and S. Nayar. A theory of specular surface geometry. *Int. J. of Computer Vision*, 1996.

[12.35] N. Page, W. Snyder, and S. Rajala. Turbine blade image processing system. In *Proc. SPIE 0397, Applications of Digital Image Processing*, volume 261, Oct 1983.

[12.36] P. Rosen and G. West. Nonparametric segmentation of curves into various representations. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(12), 1995.

[12.37] H. Schultz. Retrieving shape information from multiple image of a specular surface. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(2), February 1994.

[12.38] H. Shum, K. Ikeuchi, and R. Reddy. Principal component analysis with missing data and its application to polyhedral object modeling. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(9), 1995.

[12.39] W. Snyder and G. Bilbro. Segmentation of range images. In *Int. Conference on Robotics and Automation*, March 1985.

[12.40] B. Soroka. Generalized cylinders from parallel slices. In *Proceedings of the Conference on Pattern Recognition and Image Processing*, 1979.

[12.41] B. Soroka and R. Bajcsy. Generalized cylinders from serial sections. In *3rd Int. Joint Conf. on Pattern Recognition*, Nov 1976.

[12.42] M. Soucy and D. Laurendeau. A general surface approach to the integration of a set of range views. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(4), April 1995.

[12.43] C. Stewart. Minipran: A new robust estimator for computer vision. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(10), 1995.

[12.44] J. Stone and S. Isard. Adaptive scale filtering: A general method for obtaining shape from texture. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(7), July 1995.

[12.45] M. Subbarao and T. Choi. Accurate recovery of three-dimensional shape from image focus. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(3), March 1995.

[12.46] S. Sull and N. Ahuja. Integrated 3-d analysis and analysis-guided synthesis of flight image sequences. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(4), April 1994.

[12.47] S. Sullivan, L. Sandford, and J. Ponce. Using geometric distance fits for 3-d object modeling and recognition. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(12), 1994.

[12.48] Y. Sun, I. Liu, and J. Grady. Reconstruction of 3-d tree-like structures from three mutually orthogonal projections. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(3), March 1994.

[12.49] B. Super and A. Bovik. Shape from texture using local spectral moments. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(4), April 1995.

[12.50] K. Tarabanis, R. Tsai, and A. Kaul. Computing occlusion-free viewpoints. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(3), March 1996.

[12.51] G. Taubin. Nonplanar curve and surface estimation in 3-space. In *IEEE Robotics and Automation Conference*, May 1988.

[12.52] P. Torr and D. Murray. The development and comparison of robust methods for estimating the fundamental matrix. *International Journal of Computer Vision*, 24(3), 1997.

[12.53] E. Trucco and R. Fisher. Experiments in curvature-based segmentation of range data. *IEEE Trans. Pattern Anal. and Machine Intel.*, 17(2), 1995.

[12.54] R. Wang, A. Hanson, and E. Riseman. Fast extraction of ellipses. In *Ninth International Conference on Pattern Recognition*, 1988.

[12.55] M. Wani and B. Batchelor. Edge-region-based segmentation of range images. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(3), 1994.

[12.56] I. Weiss and M. Ray. Model-based recognition of 3d objects from single images. *IEEE Trans. Pattern Anal. and Machine Intel.*, 23(2), 2001.

[12.57] R. Woodham. Photometric method for determining surface orientation from multiple images. *Optical Engineering*, 19, 1980.

[12.58] Y. Lei and K. Wong. Ellipse detection based on symmetry. *Pattern Recognition Letters*, 20, 1999.

[12.59] X. Yu, T. Bui, and A. Kryzak. Robust estimation for range image segmentation and reconstruction. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(5), 1994.

[12.60] M. Zerroug and R. Nevatia. Three dimensional descriptions based on the analysis of the invariant and quasi-invariant properties of some curved-axis generalized cylinders. *IEEE Trans. Pattern Anal. and Machine Intel.*, 18(3), March 1996.

[12.61] R. Zhang, P. Tsai, J. Cryer, and M. Shah. Shape-from-shading: A survey. *IEEE Trans. Pattern Anal. and Machine Intel.*, 21(8), Aug 1999.

[12.62] J. Zheng. Acquiring 3-d models from sequences of contours. *IEEE Trans. Pattern Anal. and Machine Intel.*, 16(2), February 1994.

13 Developing Computer Vision Algorithms

Education is what remains after one has forgotten everything one learned in school.
– Albert Einstein

We conclude the book with a look back in the context of how one develops good algorithms to solve Computer Vision problems. The steps are listed in order, as much as possible, with reference to materials and examples in the book.

The approach is explained through an example: Suppose you need to develop an algorithm to construct a panorama from two overlapping images. This requires that you find some way to transform coordinates in one image to appropriate coordinates in the next image, as described in Chapter 12. (See Figure 13.1.)

Know the literature If you are working on a problem that you think is important, chances are somebody else has also worked on it. Since the advent of Web searching, it has become easier to search, but don't limit your searches to Google. It's OK if you spend several days looking in the library. Looking at actual paper copies of old journal papers isn't necessarily a bad thing. After all, you plan to spend months or years on this project. Some papers were written before Web searching [13.2]. Often, you can be fortunate enough to find that someone else has done a good literature survey for you. For example, in the topic of the shape from motion, [13.3] and [13.1] thoroughly teach the relevant material.

Form an Objective Function In Chapter 6, we saw one example of finding an image that is the solution to an optimization problem. We found images that resembled the input image but also had some other desirable properties. In Chapter 5 we derived convolution kernels by finding best fits to data. In Chapter 12 objective functions were explicitly used twice.

In some rare instances, it isn't possible to set up a problem in terms of an optimization problem, but it's a good way to start.

Do the Math Correctly Once you have set up an objective function, you need to minimize it. For that, you often need a gradient. Can you find the gradient analytically as in section 6.5.1, or must you use numerical methods to estimate the gradient? In Chapter 12, we solved for a homography by using two images. But even if the math is correct, is it a valid approach? Can you even use this approach to solve general stereo problems? These questions are closely related to the next topic.

Be Aware of Your Assumptions Every semester we teach this course, we encounter students who think all images are composed of pixels with brightnesses that range

Figure 13.1 Two images of the same building overlap. In fact they are the result of a simple rotation of the same camera.

between 0 and 255. A derivation that assumes nothing can be brighter than 255 can produce interesting results, such as $255 + 2 = 1$. More subtle assumptions are also around, for example, a relationship may be linear if the second camera is a rotation of the first, but not if rotations and translations are combined.

Also be aware that numerical methods such as inverse problems may be surprisingly sensitive to small errors. Learn the meaning of *ill-conditioned*. As we discussed in Chapter 6, ill-conditioned systems require special treatment.

Choose Performance Metrics In the next step, you will need to test your algorithm on some images. Wouldn't you love to say, "My algorithm works better than any other on this data!" But what do you mean *works better?* You need to decide (possibly even before you begin the work) how you will evaluate the results. What is good performance? How can you measure it? How can you compare it?

Test Your Idea on a Synthetic Image If you completed the project assignments of Chapter 5, you already simulated an ideal image. Then, you corrupted that image with blur and noise. This sort of simulation allows you to test your algorithms under the most favorable conditions. If your algorithm doesn't work on synthetic images, it almost certainly won't work on real images, which have unknown and unmeasured distortions.

Test Your Idea on a Phantom A *phantom* is a real image, but it contains features that have been very carefully measured. The lead-hole images that you may have analyzed in the homeworks of Chapter 9 are of holes drilled with high precision in a circuit board. Since you know with great precision the parameters of the object being imaged, these images represent a type of phantom, and your algorithm should return precise answers when it determines the radius of those holes.

Medical physicists calibrating x-ray machines also use a type of phantom, in the form of a block of plastic or wax with very tiny wires embedded in the wax. These wires will be visible only if the x-ray machine is working optimally.

Determine the Answer to "What Is Truth?" You probably can't find the definitive answer to this ancient question, but you can and must test your algorithm on real examples. To do that, you must determine some definition for how well your

algorithm is performing, and that requires a definition of what correct performance is. For example, in medical imaging, you might decide that if three expert radiologists agree, their collective judgement is correct by definition. You get to choose the definition, but to publish your results, you have to defend your decision.

Thoroughly Test Your Algorithm You will need to run many tests. Evaluate your algorithm as a function of multiple image distortions, such as the amount of additive noise, the diversity of test examples, etc. When it fails, learn what makes it fail.

The list of topics above isn't complete. We didn't include techniques on how to avoid letting frustration get you down.

In this book, we have introduced a few very important principles. When you encounter a real problem, you can use the principles you have mastered here to help you grasp the nature of the problem and how hard it might be to solve. Then, you can move on to more detailed literature that will provide more, but narrower direction. For example, the "bible" of multi-camera image analysis is a 400-page book [13.1], which covers in great detail the topics to which we provide an introduction in sections 12.2–12.3.

We hope that the introduction given in this book makes your life a bit easier as you move into this exciting field!

W.E.S. and H.Q.

Bibliography

[13.1] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*, volume 2. Cambridge University Press, 2000.

[13.2] Sir Isaac Newton. *Mathematical Principles of Natural Philosophy*. The Royal Society, 1687.

[13.3] M. A. Sutton, J. J. Orteu, and H. Schreier. *Image Correlation for Shape, Motion and Deformation Measurements*. Springer, 2009.

Appendix A Support Vector Machines

In Chapter 1 of this book, we distinguished between *Pattern Recognition* and *Computer Vision*. Computer Vision systems make measurements on images, and Pattern Recognition systems accept measurements as inputs and make decisions.

There is a problem with these definitions. At the time of publication of this book, many Computer Vision experts are working on the problem of *Object Recognition*, assigning a label to an object (e.g., bicycle or car) in a scene that is cluttered with other objects and background. Making such a decision is clearly a pattern recognition problem, but most research in this area is published in the Computer Vision literature.

Support Vector Machines (SVMs) are systems that accept a set of measurements from two classes, $I_1 = \{\mathbf{x}_{11}\ \mathbf{x}_{12} \dots\}$ and $I_2 = \{\mathbf{x}_{21}\ \mathbf{x}_{22} \dots\}$. Having gained information from those two sets, then given an unknown $\mathbf{x}$, the SVM can decide whether $\mathbf{x}$ belongs to class 1 or class 2. Such a machine is clearly a pattern classifier, and their operation is also taught in pattern recognition books.

Nevertheless, the SVM is often used in solving computer vision problems like object recognition, and the inputs are usually measurements from image feature measurement systems like HoG's (see section 11.6). You, the student reading this book, are likely to meet people who are working on state-of-the-art systems (or at least reading about them). Since you should know at least a bit about these systems, we will introduce you to the basic concepts.

Support Vector Machines are pattern classifiers based on the concept of minimizing structural risk, which were first introduced by Vapnik [A.7]. They provide performance superior to most other pattern classification methods when the classification problem is to distinguish between two classes.

A.1 Derivation of the Support Vector Machine

Assume the problem may be described by asking to which of two classes an unknown vector belongs, and suppose two training sets are provided, each consisting of examples from one of the classes. Further assume the two classes are separable by a linear hypersurface.

The feature space is to be divided by a hyperplane into two regions, in which the examples of the training sets are separated. Define the distance from the closest point in class 1 to the hyperplane as d_1. Similarly let d_2 be the distance from the closest point in class 2 to the hyperplane. The *margin* is defined as $d_1 + d_2$. Instead of accepting any hyperplane that divides the region space, we seek the hyperplane that maximizes the margin.

Given an unknown feature vector $\mathbf{x}$, project this sample onto some unit vector ϕ, and make a decision using the following rule:

$$\text{decide class 1 if } \phi^T\mathbf{x} - q > 0 \text{ otherwise decide class 2,} \tag{A.1}$$

where q is a scalar constant. Let $\mathbf{x}_1$ and $\mathbf{x}_2$ denote points in class 1 and class 2 respectively. Since we are seeking the points closest to the decision hyperplane, we seek $\mathbf{x}_1$ and $\mathbf{x}_2$ such that their projections onto ϕ are as close together as possible, while still being on the correct side. That is, we seek points $\mathbf{x}_1$ and $\mathbf{x}_2$ such that with minimal positive ρ,

$$\begin{aligned}\phi^T\mathbf{x_1} &= q + \rho \\ \phi^T\mathbf{x_2} &= q - \rho.\end{aligned} \tag{A.2}$$

Define I_i to be the set of training examples from class i. For any point $\mathbf{x}$ in I_i, $\phi^T\mathbf{x} - q > \rho$, and for any such point in I_2, $\phi^T\mathbf{x} - q < \rho$. We need to find two things: (1) a pair of points, one in each class, which are as close together as possible. We will call these points *support vectors*; and (2) a vector onto which to project the support vectors so that their projections are maximally far apart. We solve this problem as follows:

Recall that ϕ was a unit vector. It is thus equal to some other vector in the same direction divided by its magnitude, $\phi = \frac{\mathbf{w}}{||\mathbf{w}||}$. We will find one such vector, with certain properties that will be introduced shortly. For now, let $\mathbf{x}_1$ denote any point in I_1, not necessarily a support vector, and similarly for $\mathbf{x}_2$. Then

$$\begin{aligned}\frac{\mathbf{w}}{||\mathbf{w}||}\mathbf{x}_1 - q &\geq \rho \\ \frac{\mathbf{w}}{||\mathbf{w}||}\mathbf{x}_2 - q &\leq \rho\end{aligned} \tag{A.3}$$

which leads to

$$\begin{aligned}\mathbf{w}^T\mathbf{x}_1 - q||\mathbf{w}|| &\geq \rho||\mathbf{w}|| \\ \mathbf{w}^T\mathbf{x}_2 - q||\mathbf{w}|| &\leq \rho||\mathbf{w}||.\end{aligned} \tag{A.4}$$

Define $b = -q||\mathbf{w}||$, and constrain $\mathbf{w}$ by requiring that its magnitude satisfy:

$$||\mathbf{w}|| \ = \frac{1}{\rho}. \tag{A.5}$$

Now we have two equations that describe behavior for any points in class 1 or class 2:

$$\begin{aligned}\mathbf{w}^T\mathbf{x}_1 + b &\geq 1 \\ \mathbf{w}^T\mathbf{x}_2 + b &\leq 1.\end{aligned} \tag{A.6}$$

We seek the line that maximizes the margin, ρ, as illustrated in Figures A.1. Eq. A.5 shows that this is the same as finding the projection vector $\mathbf{w}$ whose magnitude is minimal. That is, we seek the $\mathbf{w}$ that minimizes $\mathbf{w}^T\mathbf{w}$. We have seen the derivative of a quadratic form before, and we know that the null vector would minimize this, providing a trivial and useless solution. Some constraints are needed.

Define a label y_i for the point $\mathbf{x}_i$ as

$$y_i = \begin{cases} 1 & \text{if} \quad \mathbf{x}_i \in I_1 \\ -1 & \text{if} \quad \mathbf{x}_i \in I_2. \end{cases} \tag{A.7}$$

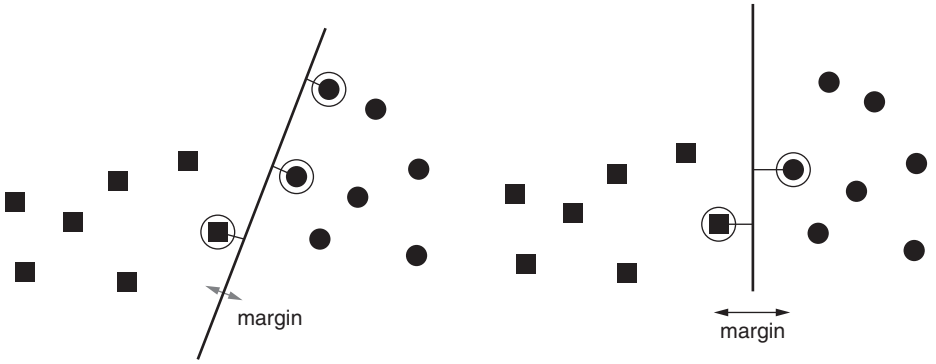


Figure A.1 LEFT: A poor choice of the dividing line is illustrated. The sets of squares and circles are linearly separable, but this line passes very close to points in each class. The support vectors are circled. RIGHT: A better choice of the dividing line is illustrated. This line passes as far as possible from any point in the two classes, producing a larger margin.

Using that definition of y_i, consider the expression $y_i(\mathbf{w}^T\mathbf{x}_i + b)$. This will always be greater than or equal to 1, regardless of the class of $\mathbf{x}_i$. This can be used as a constraint, and our minimization problem becomes: find the $\mathbf{w}$ that minimizes $\mathbf{w}^T\mathbf{w}$ such that $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$. This can be accomplished by setting up the following constrained optimization problem:

$$L = \mathbf{w}^T\mathbf{w} - \sum_{i=1}^{l} \lambda_i(\mathbf{w}^T\mathbf{x}_i + b) - 1) \tag{A.8}$$

where l is the number of the samples in the training set.

Take the partial derivative with respect to $\mathbf{w}$,

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum \lambda_i \mathbf{x}_i y_i. \tag{A.9}$$

Setting this to zero produces

$$\mathbf{w} = \sum \lambda_i \mathbf{x}_i y_i. \tag{A.10}$$

Take the derivative with respect to b,

$$\frac{\partial L}{\partial b} = -\sum \lambda_i y_i. \tag{A.11}$$

The equation for L becomes

$$L = \frac{1}{2}\sum_i \sum_j \lambda_i y_i \lambda_j y_j \mathbf{x}_i^T \mathbf{x}_j - \sum_i \sum_j \lambda_i y_i \lambda_j y_j \mathbf{x}_i^T \mathbf{x}_j - b\sum_i \lambda_i y_i + \sum_i \lambda_i. \tag{A.12}$$

The first and second terms are the same except for the 1/2. The third term is zero. Defining a matrix A by $A_{ij} = [y_i y_j \mathbf{x}_i^T \mathbf{x}_j]^T$ allows L to be written in a matrix form

$$L = -\frac{1}{2}\Lambda^T A \Lambda + \mathbf{1}^T \Lambda \tag{A.13}$$

where $\mathbf{1}$ denotes a vector of all ones.

Finding the vector of Lagrange multipliers (Λ) that minimizes L is a quadratic optimization problem. There are several numerical packages available that perform such operations. Once we have the set of Lagrange multipliers, the optimal projection vector is found by Eq. A.10, which we observe requires a summation over all the elements of the training set. To find the b that minimizes L, we need to make use of the Kuhn-Tucker [A.3] conditions:

$$\lambda_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1) = 0 \quad \text{for all } i \tag{A.14}$$

In principle, Eq. A.14 may be solved for b using any i, but it is numerically better to use an average. Similarly, we note the dimension of A is the same as the number of samples in the training set. Thus, unless some "filtering" is done on the training set prior to building the SVM, the computational complexity can be substantial.

A.2 Nonlinear Support Vector Machines

Instead of dealing with the actual samples, consider applying a nonlinear transformation that produces a vector of higher dimension, $\mathbf{y}_i = \Theta(\mathbf{x}_i)$. For example, if $\mathbf{x} = [x_1, x_2]^T$ is of dimension 2, $\mathbf{y}$ could be defined by

$$\mathbf{y} = [x_1^2\ x_2^2\ x_1x_2\ x_1\ x_2\ 1]^T \tag{A.15}$$

which is of dimension 6. This increase in dimension does not destroy the capability of the classifier. Indeed, SVMs seem to cope well with having more unknowns than samples in the training sets.

An expansion of the form described in Eq. A.15 increases the dimensionality of the space, and increases the likelihood that the classes will be linearly separable in the higher dimensional space (for reasons beyond the scope of this brief explanation). It also provides a simple mechanism for incorporating nonlinear mixtures of the information from the measurements. The polynomial form of Eq. A.15 is but one way of expanding the dimensionality of the measurement vector. A more interesting collection of ways comes to mind when one looks at Eq. A.13 and observes that to compute the optimal separating hyperplane, one does not need to know the vectors themselves, but only the scalars that result from computing all possible inner products. Thus, we do not need to map each vector to a high-dimensionality space and then take the inner product of those vectors if we can figure out ahead of time what those inner products should be.

A.3 Kernels and Inner Products

We seek the best separating hyperplane in the higher dimensional space defined by $\mathbf{y}_i = \Theta(\mathbf{x}_i)$, $\Theta : (\mathbb{R}^d \rightarrow \mathbb{R}^m)$ where $m > d$. Then, the equation for the elements of A becomes functions of the inner products of these new vectors $A = [y_iy_j\Theta^T(\mathbf{x}_i)\Theta(\mathbf{x}j)]$. For notational convenience (and to lead to a really clever result), define a kernel operator, $K(\mathbf{x}_i, \mathbf{x}_j)$, which takes into account both the nonlinear transformation, Θ, and the inner product. Instead of asking "what nonlinear operator should I use," let's ask a different question: "Given a particular kernel, is there any chance it represents the combination of a

nonlinear operator and an inner product?" The amazing answer is "yes, under certain conditions that is true." These conditions are known as Mercer's conditions: Given a kernel function $K(\mathbf{a}, \mathbf{b})$ of two vector-valued arguments, if for any $g(\mathbf{x})$ that has finite energy,[1] then $\int K(\mathbf{a}, \mathbf{b})g(\mathbf{a})g(\mathbf{b})d\mathbf{a}d\mathbf{b} \geq 0$, and there exists a mapping Φ and a decomposition of K of the form

$$K(\mathbf{a}, \mathbf{b}) = \psi_i(\mathbf{a})\psi_i(\mathbf{b}). \tag{A.16}$$

In Eq. A.16, the subscript i denotes the i^{th} element of the vector-valued function ψ. Thus, that expression represents an inner product. Notice that Mercer's conditions simply state that if K satisfies these conditions, then K may be decomposed into an inner product of two instances of a function ψ. They do not say what ψ is, nor do they say what the dimensionality of ψ is. But that's OK. We do not have to know. In fact, the vector ψ may have infinite dimensionality. That's still OK.

One kernel that is known to satisfy Mercer's conditions and that is very popular in the SVM literature is the radial basis function

$$K(\mathbf{a}, \mathbf{b}) = \exp\left(-\frac{(\mathbf{a}-\mathbf{b})^T(\mathbf{a}-\mathbf{b})}{2\sigma^2}\right). \tag{A.17}$$

In the literature, Support Vector Machines have been applied to problems such as face recognition [A.6], bird recognition [A.10], general object recognition [A.2] breast cancer detection [A.1], and many others. In comparative analyses in the literature, they have empirically been shown to outperform classical classification tools such as neural networks and nearest neighbor rules [A.5, A.8, A.9]. Interestingly, in a comparison with a classifier based on hyperspectral data, an SVM-based classifier using multispectral data (derived from the original hyperspectral data by filtering) performed better than classifiers based on the original data [A.4].

Bibliography

[A.1] P. S. Bradley, U. M. Fayyad, and O. L. Mangasarian. Mathematical programming for data mining: Formulations and challenges. *INFORMS Journal on Computing*, 11(3), 1999.

[A.2] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE Trans Pattern Anal Mach Intell.*, 32(9), 2010.

[A.3] R. Fletcher. *Practical Methods of Optimization*. Wiley, 1987.

[A.4] B. Karaçali and W. Snyder. On-the-fly multispectral automatic target recognition. In *Combat Identification Systems Conference*, Jun 2002.

[A.5] D. Li, S. M. R. Azimi, and D. J. Dobeck. Comparison of different neural network classification paradigms for underwater target discrimination. In *Proceedings of SPIE, Detection and Remediation Technologies for Mines and Minelike Targets V*, volume 4038, pages 334–345, 2000.

[A.6] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection. In *Proceedings of CVPR'97*, Jun 1997.

[A.7] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[1] $\int (g_2(\mathbf{x})d\mathbf{x}$ is finite).

[A.8] M. H. Yang and B. Moghaddam. Gender classification using support vector machines. In *Proceedings of IEEE International Conference on Image Processing*, volume 2, pages 471–474, 2000.

[A.9] Y. Yang and X. Liu. Re-examination of text categorization methods. In *Proceedings of the 1999 22nd International Conference on Research and Development in Information Retrieval (SIGIR'99)*, pages 42–49, 1999.

[A.10] N. Zhang, R. Farrell, F. Iandola, and T. Darrell. Deformable part descriptors for fine-grained recognition and attribute prediction. In *IEEE International Conference on Computer Vision*, 2013.

Appendix B How to Differentiate a Function Containing a Kernel Operator

The function of Eq. 6.25 is complicated by involving a sum over all the pixels and by the use of convolution that creates terms involving more than one pixel at a time. This Appendix discusses the differentiation of such an expression. The material is placed in an appendix because it is not essential to understanding the concepts of noise removal, but is important for those who must implement software.

In previous discussions, the effects of blur were excluded and noise was assumed to be the only source of distortion. In this section, the results derived in chapter 6 are extended and include blur in the formation of the measured image g, such that the noise term can be written as Eq. B.1. We will illustrate differentiation of the prior and the noise terms as if both contained convolutions. First, we illustrate how to differentiate a function containing a kernel operator (or a blur operator).

$$\sum_i ((f \otimes h)_i - g_i)^2 \tag{B.1}$$

Taking a one-dimensional example, assume f_i is a pixel from the original (unknown) image, g_i is a pixel from the measured image, and h is the horizontal blur kernel with a finite kernel size 5, as shown below:

.....	f_2	f_3	f_4	f_5	f_6	.....

h_{-2}	h_{-1}	h_0	h_1	h_2

Here, the derivative of the noise term with respect to f in Eq. B.1, which is required in order to use gradient descent, is examined in detail. We will make use of the fact that we are taking a partial derivative of a summation. Therefore, any term in the sum that does not involve the denominator of the partial derivative will be zero.

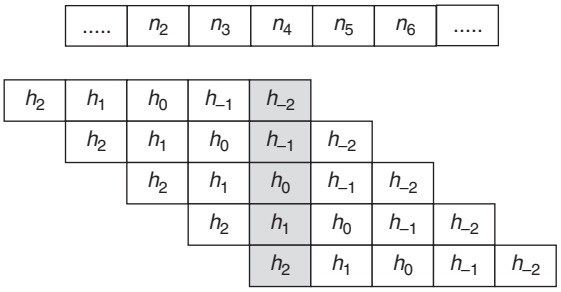


Figure B.1 The reverse kernel in the derivation of the noise term.

First write out all the terms involving a pixel (let's pick f_4) at which a measurement (g_4) was made in the noise term H_n above

$$\begin{aligned} E_4 &= ((f \otimes h)_2 - g_2)^2 + ((f \otimes h)_3 - g_3)^2 + ((f \otimes h)_4 - g_4)^2 \\ &\quad + ((f \otimes h)_5 - g_5)^2 + ((f \otimes h)_6 - g_6)^2 \\ &= (f_0 h_{-2} + f_1 h_{-1} + f_2 h_0 + f_3 h_1 + f_4 h_2 - g_2)^2 \\ &\quad + (f_1 h_{-2} + f_2 h_{-1} + f_3 h_0 + f_4 h_1 + f_5 h_2 - g_3)^2 \\ &\quad + (f_2 h_{-2} + f_3 h_{-1} + f_4 h_0 + f_5 h_1 + f_6 h_2 - g_4)^2 \\ &\quad + (f_3 h_{-2} + f_4 h_{-1} + f_5 h_0 + f_6 h_1 + f_7 h_2 - g_5)^2 \\ &\quad + (f_4 h_{-2} + f_5 h_{-1} + f_6 h_0 + f_7 h_1 + f_8 h_2 - g_6)^2 \end{aligned} \tag{B.2}$$

where $(f \otimes h)_i$ denotes the application of kernel h to image f with the origin of h (in this case, the center) located at pixel f_i.

The derivative of H_n with respect to pixel f_4 can then be derived as:

$$\begin{aligned} \frac{\partial H_n}{\partial f_4} &= 2((f \otimes h)_2 - g_2)h_2 + 2((f \otimes h)_3 - g_3)h_1 + 2((f \otimes h)_4 - g_4)h_0 \\ &\quad + 2((f \otimes h)_5 - g_5)h_{-1} + 2((f \otimes h)_6 - g_6)h_{-2} \end{aligned} \tag{B.3}$$

$$\frac{\partial H_n}{\partial f_4} = (((f \otimes h) - g) \otimes h_{rev})|_{f_4} \tag{B.4}$$

where the reverse of the kernel h, $h_{rev} = \boxed{h_2}\boxed{h_1}\boxed{h_0}\boxed{h_{-1}}\boxed{h_{-2}}$, and $(f \otimes h - g)$ is computed at all points. The application of h_{rev} is illustrated more clearly in Figure B.1 where we assume $n_i = ((f \otimes h) - g)_i$. The application of the reverse kernel with a two-dimensional image follows the same rule as that in the one-dimensional case. Equation B.5 shows a 3×3 kernel function (h) and the corresponding reverse kernel h_{rev}.

$$h = \begin{bmatrix} h_{-1,-1} & h_{-1,0} & h_{-1,1} \\ h_{0,-1} & h_{0,0} & h_{0,1} \\ h_{1,-1} & h_{1,0} & h_{1,1} \end{bmatrix}, \quad h_{rev} = \begin{bmatrix} h_{1,1} & h_{1,0} & h_{1,-1} \\ h_{0,1} & h_{0,0} & h_{0,-1} \\ h_{-1,1} & h_{-1,0} & h_{-1,-1} \end{bmatrix} \tag{B.5}$$

Thus, the general form of differentiation when a kernel function is involved is as follows: Let $\Lambda(f \otimes h)$ be some differentiable function. The derivative with respect to f is

$$\frac{\partial}{\partial f}\Lambda(f \otimes h) = (\Lambda'(f \otimes h)) \otimes h_{rev} \tag{B.6}$$

where $\Lambda'(\xi) = \frac{\partial}{\partial \xi}\Lambda(\xi)$.

Besides the noise term, the prior term can also contain a kernel operator. Recall that the prior energy function models the neighborhood operation and thus can be represented as a kernel operation with the kernel selected dependent on the properties of the image. We illustrate the gradient using the following prior:

$$H_p = -\sum_i \exp(-(f \otimes r)^2) \tag{B.7}$$

$$\frac{\partial H_p}{\partial f} = [2(f \otimes r)\exp(-(f \otimes r)^2)] \otimes r_{rev}. \tag{B.8}$$

Recall that the derivative of the prior, $\frac{\partial H_p}{\partial f}$, is itself an image, and that image is derived by applying r to f, multiplying (pixel by pixel) with the exponential to produce another image, and applying the reverse of r to that image.

Author Index

Subject Index